Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Arquitetura de Gerenciamento de Identidades Usando OpenID e Cartões Inteligentes

Autor:

Pedro Henrique Valverde Guimarães

Orientador:

Prof. Otto Carlos Muniz Bandeira Duarte, Dr. Ing.

Examinador:

Prof. Luís Henrique Maciel Kosmalski Costa, Dr.

Examinador:

Prof. Aloysio de Castro Pinto Pedroza, Dr.

DEL

Novembro de 2012

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

À minha família.

AGRADECIMENTO

Agradeço a minha família, sem a qual não seria possível chegar até aqui. Em especial, agradeço aos meus pais e irmão, que sempre estiveram lá para me dar suporte e estímulo para completar um passo tão importante na minha vida.

Agradeço a todos que participaram da minha graduação, colegas de turma e de trabalho e os professores que me formaram. Agradeço a equipe do Grupo de Teleinformática e Automação, que me apoiou e incentivou durante todas as etapas desse trabalho. Em especial, agradeço a meu orientador, o professor Otto Duarte, cujos conselhos, dedicação, paciência e diálogos foram fundamentais para esse projeto. Agradeço também a banca que participou da avaliação desse projeto final de curso pela atenção.

Agradeço a UFRJ e a Télécom ParisTech por me receberem como aluno e abrirem portas para o meu futuro profissional. Também agradeço a todas as pessoas que participaram da minha vida durante esse tempo como estudante de graduação, elas transformaram tudo isso em uma experiência de vida rica e completa.

Por fim, agradeço a FINEP, CNPq, CAPES, UOL e FAPERJ pelo financiamento deste trabalho.

RESUMO

O gerenciamento de identidades representa um grande desafio na Internet. Atualmente, cada serviço tem o seu próprio sistema de identificação e um usuário é obrigado a memorizar muitos apelidos e senhas diferentes para acessá-los. O acúmulo de apelidos e senhas pode levar a uma séria falha em segurança, pois os usuários costumam criar senhas fracas ou repeti-las para facilitar a memorização. A grande dificuldade é criar um sistema de gerenciamento capaz de proteger a identidade digital de seus usuários, evitando o roubo ou perda de dados. Há diversas propostas que tentam solucionar esse problema. O serviço de gerenciamento de identidades na Internet oferece a facilidade do usuário ter apenas um apelido e uma única senha e a partir deste serviço conseguir acessar todos os outros serviços. Este trabalho propõe uma solução baseada em uma especificação conhecida como OpenID. O OpenID permite uma comunicação simples e segura entre os usuários, os provedores de serviços e de identidades. A implementação realizada se baseia no uso de um tipo de microcontroladores seguros, os cartões inteligentes - *smart cards*, para a autenticação de usuários de forma confiável, o que garante um alto nível de segurança. Dessa forma, um usuário não precisa mais memorizar diversas senhas, pois esses microcontroladores podem se comunicar com o provedor de identidades de forma muito mais segura. A segurança da implementação desse projeto foi analisada de acordo com ataques conhecidos ao OpenID. Essa análise mostra que o uso do OpenID com cartões inteligentes melhora a segurança geral do sistema, principalmente por evitar o roubo de credenciais por vírus no computador, *phishing* e *keylogging*. Deve ser destacado que a melhora de segurança é obtida sem aumentar a complexidade do processo de autenticação e permitindo que qualquer usuário possa utiliza-lo.

Palavras-Chave: sistema de gerenciamento de identidade, OpenID, smart card, microcontrolador seguro, autenticação mútua.

ABSTRACT

The identity management represents a big issue on the Internet. Nowadays, each service has its own identification system and users are obliged to have in mind many nicknames and passwords to access them. The increasing number of nicknames and passwords may lead to serious security flaws, because these passwords are weaker or repeated to ease memorization. The main difficulty is to create a management system capable of protecting users' digital identity, preventing theft or loss of data. There are various proposals that try to solve this issue. The service of identity management on the Internet eases users' life by allowing them to have one password to access all the services. This work proposes a solution based on protocol standard known as OpenID. The OpenID allows a safe and simple communication between the users, service and identity providers. The implementation assures a high-level security because it is based on one type of secure microcontrollers, known as smart cards, for a reliable way for users' authentication. This way, users do not need to memorize passwords since microcontrollers can securely communicate with identity providers. The implementation security was analyzed for this project, according to known attacks to the OpenID. The analysis shows that the usage of the OpenID with smart cards can improve the overall security of the system and prevent credential stealing through computer viruses, phishing or keylogging. The improvement on the overall security does not increase the complexity of the authentication process and allowing any type of user to manage it.

Key-words: identity management system, OpenID, smart card, secure microcontroller, mutual authentication.

Siglas

AID *Application Identifier* – Identificador de Aplicação no cartão Java

AIK *Attestation Identity Keys*

APDU *Application Protocol Data Unit* - Unidade de Dado do Protocolo de Aplicação

CLA *Class Byte* - *Byte* de Classe da APDU de Comando

CRSF *Cross-Site-Request-Forgery* - Ataque de Requisição Cruzada de Sessões

DH *Diffie-Hellman Key Exchange* - Troca de Chaves Diffie-Hellman

EPAL *Enterprise Privacy Authorization Language*

FIPS *Federal Information Processing Standard* - Orgão de Padronização do governo dos EUA para sistemas de computadores

FITS *Future Internet Testbed with Security* – Testebed de Internet do Futuro com Segurança

HMAC *Keyed-Hashing Message Authentication Code*

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

ICC *Integrated Circuit Card* - Cartão com Circuito Integrado

IdP *Identity Provider* - Provedor de Identidades

INS *Instruction Byte* - *Byte* de Instrução da APDU de Comando

IOI *Itens of Interest* – Itens de Interesse

ITU *International Telecommunication Union*

JCRE *Java Card Runtime Environment* - Ambiente Java Card em Tempo de Execução

JCVM *Java Card Virtual Machine* - Máquina Virtual para Cartões Java

Lc *Length Command Byte* - *Byte* da APDU de comando que define a quantidade de *bytes* enviados nesse comando

Le *Length Expected Byte* - *Byte* da APDU de comando que define a quantidade de *bytes* esperados na resposta

MVC *Model-View-Controller* – Padrão para projeto de programas de computador chamado Modelo-Interface-Controle

OP *OpenID Provider* - Provedor de Identidades OpenID

P1 *Parameter Byte 1* - Primeiro *Byte* de Parâmetro para a Instrução da APDU de Comando

P2 *Parameter Byte 2* - Segundo *Byte* de Parâmetro para a Instrução da APDU de Comando

P3P *Platform for Privacy Preferences*

PAD *Personal Authentication Device* – Aparelho Pessoal de Autenticação

SAML *Security Assertion Markup Language*

SHA *Secure Hash Algothim* - Algoritmo Seguro de *Hash*

SLO *Single Logout* – Encerramento Único de Sessões

SOAP *Simple Object Access Protocol*

SP *Relying party* - Provedor de Serviços no OpenID

SP *Service Provider* - Provedor de Serviços

SSL *Secure Socket Layer*

SSO *Single Sign-On* – Autenticação Mútua

SW *Status Word* - 2 *bytes* que retornam o status de encerramento do processamento do comando, 0x9000 representa encerramento normal

TLS *Transport Layer Security*

TPM *Trusted Platform Module*

UFRJ Universidade Federal do Rio de Janeiro

URI *Uniform Resource Identifier* - Identificador-Padrão de Recursos

URL *Uniform Resource Locator* - Localizador-Padrão de Recursos

XACML *eXtensible Control Markup Language*

XRDS *eXtensible Resource Descriptor Sequence*

XRI *eXtensible Resource Identifier* - Identificador de Recursos eXtensível

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 O Gerenciamento de Identidades

O gerenciamento de identidades na Internet é uma questão fundamental e atualmente sem soluções simples. A segurança *online* depende fortemente de identificar de forma correta uma determinada entidade ou usuário. O problema a ser resolvido é de garantir a segurança e simplicidade do procedimento de identificação de usuários que desejam acessar serviços na Internet.

Atualmente, provedores de serviços exigem que seus usuários se registrem antes que possam usar um serviço. O usuário, por sua vez, precisa se registrar em uma grande variedade de sítios da Internet que oferecem serviços e é obrigado a memorizar uma quantidade enorme de credenciais de autenticação (geralmente, apelidos e senhas), sendo comum esquecer senhas ou criar senhas fracas, suscetíveis a ataques como de força bruta ou de dicionário.

Esse projeto se baseia na proposta de transformar o gerenciamento de identidades em um serviço na Internet, no qual um usuário pode se registrar em um único provedor e utilizá-lo para acessar qualquer serviço. Esses provedores especiais são conhecidos como provedores de identidades (*Identity Providers* - IdP). Nesse novo modelo de identificação existe uma entidade responsável por confirmar se um usuário é ou não quem alega ser. Os provedores de serviços por sua vez confiam que essa entidade é capaz de autenticar e retornar o resultado de forma segura.

1.2 A Proposta do Projeto

O objeto de estudo desse trabalho é propor e implementar um sistema de gerenciamento de identidades capaz de autenticar seus usuários de forma simples e segura para serviços na Internet. A simplicidade e forte segurança do sistema proposto são baseadas no emprego do cartão inteligente (*smart card*). O cartão inteligente faz uso de um microcontrolador seguro, que é considerado inviolável. O uso do cartão inteligente em substituição ao apelido e senha tem também a vantagem de eliminar a possibilidade de roubo de credenciais, uma vez que ele é capaz de executar algoritmos de criptografia e guardar chaves em seu interior, o que permite uma segurança ainda maior que o conhecido apelido e senha. O cartão inteligente também é simples de se usar, pois não requer nenhuma habilidade ou conhecimento adicional do usuário. O projeto diz respeito a qualquer provedor de serviços que delegue a função de autenticar seus usuários a sistemas de gerenciamento de identidades.

1.3 Justificativa e Objetivos

O objetivo desse projeto é definir e implementar um procedimento de autenticação única suficientemente seguro para os padrões atuais da Internet. Com esse intuito, protocolos de comunicação entre o usuário e os provedores de serviço e identidade disponíveis foram estudados, assim como técnicas de autenticação seguras.

A principal motivação para que o procedimento de identificação de usuários seja repensado é a segurança. O sistema atual sobrecarrega o usuário uma vez que ele deve preencher diversos formulários de registro e deve também memorizar diversas senhas e apelidos diferentes, para diferentes serviços. Este acúmulo de apelidos e senhas a serem memorizadas acaba levando o usuário a criar senhas fracas e até repeti-las, tornando frágeis os procedimentos de autenticação atualmente disponíveis. Em contrapartida, sistemas de gerenciamento de identidades podem resolver esse problema, pois oferecem a Autenticação Única (*Single Sign-On*). Assim, o usuário precisa apenas se autenticar em um único provedor, o seu provedor de identidades, para poder acessar os serviços em que está registrado.

Tão importante quanto resolver os problemas de preenchimento de inúmeros formulários, de memorização de múltiplas senhas e de senhas fracas, os sistemas de

gerenciamento de identidades devem garantir maior segurança nos procedimentos de autenticação. Para isso, outras formas de autenticação além do conhecido esquema de apelido e senha vão ser abordados. Além disso, o uso do cartão inteligente faz com que as informações guardadas no seu interior fiquem invioláveis a ameaças, como por exemplo *keyloggers* para registrar as teclas digitadas e o armazenamento da senha em claro na memória do computador, susceptíveis a ataques de vírus.

A implementação realizada consiste de um sistema de gerenciamento de identidades usando o protocolo OpenID com a autenticação feita por cartões inteligentes (*smart cards*). O OpenID define um padrão de mensagens HTTP (*Hypertext Transfer Protocol*) trocados entre usuários, provedores de serviços e identidades. Cartões inteligentes por sua vez garantem maior segurança na autenticação, uma vez que podem agir como repositórios confiáveis de dados e conseguem executar operações de criptografia dentro do cartão. Isso garante que apenas o provedor de identidades e o cartão consigam decriptografar as mensagens trocadas.

1.4 Metodologia

Esse trabalho consiste no projeto e na implementação de um sistema de gerenciamento de identidades usando cartões inteligentes. O sistema possibilita o procedimento de autenticação única enquanto o cartão utiliza algoritmos de criptografia para garantir uma comunicação fortemente segura.

Cartões inteligentes são seguros para guardar e processar dados. Esses cartões possuem unidades de memória capazes de proteger os dados de qualquer adulteração ou leitura indevida. Além disso, estes cartões possuem processadores especiais para criptografia. Nesse projeto eles vão ser empregados em uma comunicação segura direta entre o sistema de gerenciamento de identidades e o cartão em si. O protocolo de autenticação é baseado em desafios (*challenges*), ou seja, números gerados aleatoriamente e criptografados de tal forma que apenas alguém que tenha a chave privada correta possa decriptografar. Esse protocolo foi definido para prover autenticação mútua, de tal forma que o cartão possa provar a sua identidade assim como o servidor possa comprovar a sua, evitando ataques de *phishing*, que hoje é um grande problema na Internet.

A primeira parte desse projeto foi o estudo sobre sistemas de gerenciamento

de identidades e seus protocolos. A segunda parte foi a implementação de um provedor de identidades capaz de autenticar remotamente um usuário através de seu cartão. Para isso, um protocolo de autenticação foi especificado.

1.5 Organização do Texto

Esse texto pode ser dividido em duas grandes partes, o estudo de temas relacionados e o projeto em si. A primeira parte se dedica ao estudo de gerenciamento de identidades e de cartões inteligentes. A segunda parte foca no desenvolvimento de um sistema de gerenciamento de identidades utilizando cartões inteligentes durante o procedimento de autenticação. Finalmente, discute-se a implementação do projeto, resultados, conclusão e trabalhos futuros.

O Capítulo 2 aborda o conceito de identidade na Internet. Define-se o que é a identidade digital, de acordo com algumas das referências, e, em seguida, descrevem-se as arquiteturas de gerenciamento de identidades.

O Capítulo 3 é dedicado ao OpenID. Essa especificação foi a selecionada para implementar o sistema de gerenciamento de identidades desse trabalho. Sua escolha se deveu a sua simplicidade, tanto para o provedor de serviços, já que todas as mensagens trocadas são mensagens HTTP, quanto para o usuário, que não precisa de nenhum *software* especial além de seu navegador.

O Capítulo 4 aborda o uso de cartões inteligentes. Primeiro estudando o funcionamento desses cartões em geral. Em seguida, discutem-se os protocolos de comunicação entre cartões e os terminais no qual eles se conectam. O final do capítulo é dedicado a um tipo específico de cartões, os *java cards*. Esses cartões possuem máquinas virtuais java implementadas, o que permite uma grande versatilidade ao programá-los.

O Capítulo 5 fala sobre o trabalho feito para se unir o OpenID com smart cards. O capítulo começa por relacionar trabalhos já feitos sobre o assunto, descrever os protocolos desenvolvidos para registrar e autenticar usuários e finalmente descreve a arquitetura da implementação do servidor.

O Capítulo 6 descreve os resultados obtidos.

O Capítulo 7 descreve conclusões e discute melhorias que podem ser feitas, tanto para a implementação, quanto no próprio OpenID.

Capítulo 2

A Internet e o Gerenciamento de Identidades

Há diversas definições para Identidade Digital. Identidade, de acordo com o dicionário Michaelis significa: "conjunto dos caracteres próprios de uma pessoa, tais como nome, profissão, sexo, impressões digitais, defeitos físicos etc., o qual é considerado exclusivo dela e, consequentemente, considerado, quando ela precisa ser reconhecida". Na Internet essa definição se torna confusa, uma vez que esta definição não foi pensada para que o usuário saiba com quem ou o que está se conectando. Com o crescimento do uso comercial da Internet, a identificação segura se tornou um problema ainda mais crítico.

De acordo com Bertino e Takahashi [1], a identidade digital pode ser definida como a representação de toda a informação disponível sobre um determinado indivíduo ou organização na Internet. Essa informação pode ter diferentes usos, como por exemplo, para definir permissões de acesso, dados guardados por serviços, além de oferecer ao usuário uma experiência personalizada. De acordo com a *International Telecommunication Union* [2], uma identidade pode ser dividida em três tipos de informações:

- identificador: um conjunto de informações que identifica de forma única uma entidade, como por exemplo, o RG para pessoas ou CNPJ para empresas;
- credencial: um objeto que uma entidade usa para provar que é quem alega ser, a autenticação, como por exemplo, a senha de acesso a um serviço;
- atributos: informação descritiva sobre uma entidade, que a especifica ou caracteriza, como por exemplo a altura, o sexo, a cor dos olhos de uma pessoa

ou ramo de negócios de uma empresa.

Na Internet, as identidades digitais encontram-se geralmente "pulverizadas" entre diferentes serviços. Uma única entidade ou usuário no mundo real tem várias representações digitais. A centralização de todos os dados que compõem as identidades digitais é praticamente impossível, pois há interesse dos provedores de serviços fornecerem identidades próprias a seus usuários, tanto por questões de segurança quanto para ter posse de mais informações sobre esses usuários. Mas mesmo que todos os provedores chegassem a um acordo, ainda é uma incógnita a forma de unificar o conceito de identidade digital em todos os países conectados à Internet [3].

2.1 Discussão sobre Acesso a Identidade

Para garantir acesso a um determinado serviço, geralmente o usuário deve provar que ele é quem alega ser. Essa prova denomina-se autenticação. Os procedimentos de autenticação podem ser de três tipos:

- o usuário prova ser ele mesmo, pois conhece um segredo que somente o usuário autêntico e o provedor de serviços conheçam, como por exemplo, apelido e senha;
- o usuário possui algo que somente o usuário autêntico possui, como por exemplo, um certificado assinado ou um cartão inteligente;
- o usuário possui algo em si que o identifica de forma única, como por exemplo, impressões digitais, formato da íris, vasos sanguíneos da mão, entre outras formas biométricas.

Atualmente, os procedimentos de autenticação mais comuns são compostos pelo esquema apelido e senha (o que ele sabe), obrigando o usuário a memorizar uma nova senha para cada serviço. Com o aumento de serviços, os usuários são forçados a ter na memória uma quantidade de senhas muito grande. Isso leva a três problemas principais:

- esquecer senhas: possível perda dos seus dados em um determinado serviço;
- criar senhas fracas: um usuário facilita invasões, o que compromete seriamente a sua privacidade;

- repetir senhas: usar a mesma senha para diversos serviços o que muitas vezes faz com que uma senha descoberta por engenharia social posse ser usada para acessar serviços importantes.

Uma solução para esse problema seria oferecer o gerenciamento de identidades como um serviço a parte na Internet. Nesse provedor de identidades, o usuário se registra e cadastra os seus dados uma única vez. Quando desejar acessar um determinado serviço, o provedor de serviços verifica se a identidade apresentada é válida com o provedor de identidades e requisita os dados necessários. Essa verificação obriga que o usuário se autentique no seu provedor de identidades. Como essa autenticação ocorre em um único lugar e pode ser usada para vários serviços, chama-se esse procedimento de Autenticação Única (*Single Sign-On* - SSO). O usuário não é mais obrigado a memorizar uma série de senhas e fazer vários cadastros. Os provedores de serviços se livram da incumbência de gerenciar a autenticação. Porém, esse procedimento possui uma falha séria: se um atacante roubar as credenciais de um usuário, ele poderá acessar quaisquer serviços deste usuário. Portanto, para que esse novo sistema funcione, é necessário utilizar um procedimento de autenticação mais seguro.

2.2 Redes Colaborativas

A Internet permite que seus usuários interajam com uma grande variedade de serviços de forma rápida e intuitiva. Sendo comum um usuário ter cadastro em vários serviços na Web e de haver uma interconexão entre serviços. De outro lado, Provedores de Serviços (*Service Providers* – SP) têm um interesse cada vez maior em compartilhar informações. A segurança é fundamental para garantir a confiança dessas entidades na rede. Por sua vez, a segurança esbarra em como autenticar e autorizar corretamente seus usuários. Um exemplo desse caso é um usuário que deseja reservar uma viagem: ele precisa comprar suas passagens e reservar um hotel. Nesse cenário, o usuário teria que acessar pelo menos dois provedores diferentes, se cadastrar ou autenticar neles para ter acesso aos dados. Uma solução interessante seria um sistema capaz de cadastrar e autenticar esse usuário em ambos os serviços uma única vez e de forma segura [4].

A ideia de unificar os procedimentos de autorização e autenticação dos en-

volvidos nessas redes leva à necessidade do gerenciamento de identidades entre os diferentes provedores de serviço (SPs).

2.3 Sistemas de Gerenciamento de Identidades

Um Sistema de Gerenciamento de Identidades é capaz de comprovar de forma segura que um determinado usuário possui a identidade que alega ter e comunicar isso aos provedores de serviços de acordo com as políticas de uso estabelecidas. Atualmente, esse gerenciamento é regido por cada serviço e a comunicação entre os diferentes provedores é pequena, o que obriga o usuário a ter um cadastro de seus dados pessoais e uma senha para cada serviço.

De acordo com Josang e Pope [5], podem ser citados quatro modelos de Gerenciamento de Identidades:

- Isolado: Padrão atualmente adotado pela maioria dos serviços na Internet, no qual provedores de serviços exercem também a tarefa de identificação de usuários;

- Federado: Um grupo de provedores de serviços define acordos que permitam mapear as identidades específicas de um provedor a outro;

- Centralizado: Nesse modelo existe um único provedor de identidades em que todos os provedores de serviços confiam. Esse provedor de identidades pode ser implementado de diversas formas;

- Centrado no Usuário: Usuários possuem Aparelhos Pessoais de Autenticação (*Personal Authentication Devices* - PADs). Quando um usuário deseja se autenticar em um provedor de serviços, ele apresenta seu PAD e é autenticado. Existem várias arquiteturas possíveis, onde um PAD pode carregar as credenciais dos serviços do usuário até um aparelho que grave os dados do usuário e transmita essa informação durante a autenticação.

Sistemas de Gerenciamento de Identidade Centralizados preveem a separação entre a identificação de um usuário e os serviços. Essa nova política tem diversas vantagens: o usuário precisa se cadastrar apenas junto ao Provedor de Identidade (*Identity Provider* – IdP), inserindo uma única vez suas informações pessoais e os

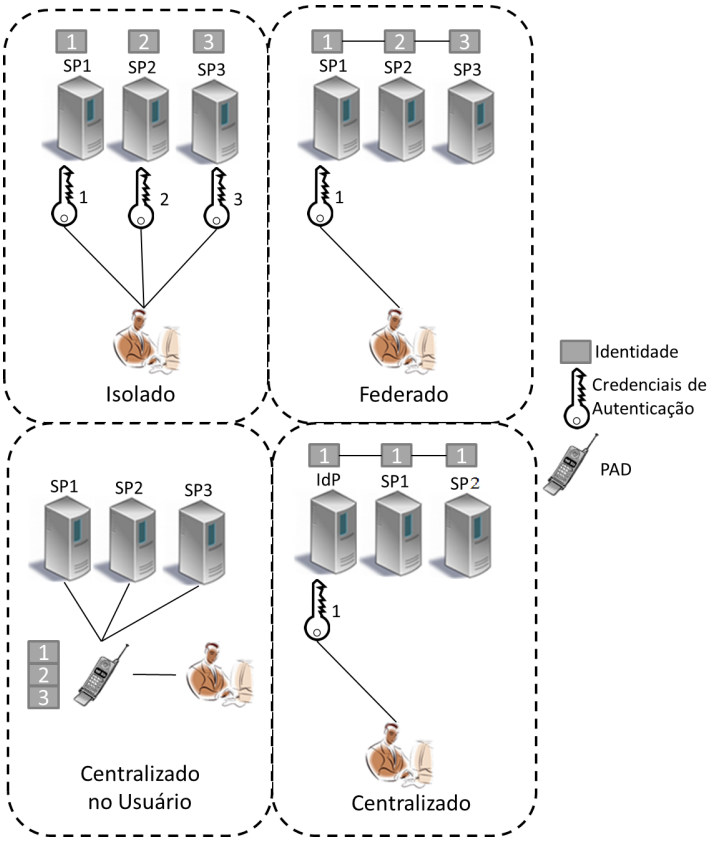

Figura 2.1: Os Modelos de Gerenciamento de Identidade: isolado, federado, centralizado e centrado no usuário.

Provedores de Serviços (*Service Provider* – SP) não precisam guardar informações específicas para identificá-los. Um usuário que queira se autenticar em um serviço deverá se autenticar previamente em seu IdP. O acesso aos seus dados pessoais pode também ser controlado via IdP, onde o usuário define quais informações cada serviço poderá acessar. Uma vez identificado, todos os serviços que aceitem aquele IdP como válido não exigirão que o usuário se autentique novamente.

A Autenticação Única já é usada por grandes provedores, que contam com muitos serviços diferentes. Alguns exemplos são a Google, com seu *Google Account* ou o *Facebook Connect*. Mas a Identidade Digital de cada usuário ainda encontra-se separada em cada um desses Provedores. O Sistema de Gerenciamento é um provedor completamente independente, onde uma vez que um usuário provou possuir uma determinada identidade, ele terá acesso a todos os serviços que essa identidade tem direito, o que permite aos usuários terem maior controle sobre o fluxo de seus dados pessoais.

2.3.1 Novos Desafios

O uso de um Sistema de Gerenciamento de Identidades pode reduzir os custos e a responsabilidade de provedores de serviços, onde o provedor de identidades (IdP) é responsável por um procedimento de autenticação que seja suficientemente seguro. Porém, inserir um novo participante nesse procedimento cria outro ponto fraco e a perda de uma identidade em um sistema de autenticação única pode causar danos bem maiores aos usuários. Além do problema de segurança, existem outros desafios, os principais citados em [6] são:

- os Sistemas de Gerenciamento de Identidade devem ganhar a confiança tanto dos usuários quanto dos Provedores de Serviços;

- necessidade de autenticação mútua: os Provedores de Serviços devem redirecionar os usuários para os provedores de identidade(IdPs), e vice-versa após o procedimento de autenticação. Para evitar que o usuário seja redirecionado para um site falso e tenha as suas informações roubadas (*phishing*), ele deve ser capaz de autenticar tanto o Provedor de Serviços quanto os Gerenciadores de Identidade;

- gerenciar a sua identidade quase nunca é um objetivo final para o usuário, mesmo que este ofereça maior segurança e privacidade. Os usuários não estão dispostos a investir muito tempo nem dinheiro nessas melhorias;

- os Provedores de Serviços querem controlar a experiência do usuário na Internet para monitorar o uso de seus serviços, evitar abusos ou proteger os dados de seus clientes.

As principais vantagens que esse novo conceito trazem são: o usuário não precisa mais memorizar diversas senhas, os Provedores de Serviços se livram da responsabilidade de gerenciar as informações necessárias para a autenticação e os Provedores de Identidade podem focar em oferecer sistemas de autenticação mais seguros que o uso de senhas [7].

Um Sistema de Gerenciamento de Identidades deve considerar as desvantagens citadas. Primeiramente, criando um sistema de autenticação mais seguro que o atual, usando um protocolo para Autenticação Única, ganhando assim a confiança dos usuários e facilitando a implementação de provedores de serviços. O sistema também deve oferecer uma forma de cadastramento simples e permitir ao usuário maior controle sobre os seus dados e quais provedores podem visualizá-los, caso deseje.

2.3.2 Privacidade

Privacidade possui diversas definições. De acordo com Pftizmann e Hansen [8], privacidade pode ser dividida em vários pontos. Os principais são: anonimato, não-associabilidade e pseudônimos. a garantia do anonimato em uma rede de comunicação pode ser definida como: "Anonimato de uma entidade é tornar essa entidade não identificável entre várias outras entidades, conhecido como *conjunto de anonimato*". Para isso, não apenas o identificador do usuário deve ser protegido, como quaisquer dados que possam ligá-lo a sua identidade. Esses dados são conhecidos como *Itens of Interest* (IOIs) e a propriedade de não ligar esses dados à identidade é conhecida como "não associabilidade" (*unlinkability*). Tornar IOIs, usuários e os destinos indistinguíveis é a condição mínima para o anonimato. Existem mecanismos para garantir isso, como por exemplo, o uso e criptografia [8].

Atingir uma comunicação completamente anônima entre um usuário e seus serviços é atualmente impossível, pois muitos desses serviços tratam de forma personalizada cada um. Uma solução é utilizar pseudônimos para usuários. Um pseudônimo é um identificador que não precisa ter necessariamente ligação direta com alguma informação desse usuário, como por exemplo, o nome completo. Canais seguros como o *Secure Sockets Layer* (SSL) ou *Transport Layer Security* (TLS) são outra forma de garantir anonimato para um possível atacante que possa estar monitorando o tráfego. Esses canais garantem que as mensagens trocadas sejam criptografadas para qualquer um que as observe.

A última questão é como garantir que um serviço não utilize as informações concedidas por usuários de forma imprópria. Essa questão é complexa em redes de comunicação e um grande problema para a Internet, uma vez que exige usuários conscientes da forma como gerenciar seus dados pessoais e os serviços devem estabelecer regras claras de como esses dados vão ser manipulados ou divulgados. Muitos trabalhos foram feitos sobre essa questão. No caso da Internet, algumas especificações foram propostas como a P3P (*Platform for Privacy Preferences*), EPAL (*Enterprise Privacy Authorization Language*) ou a XACML (*eXtensible Control Markup Language*). Essas especificações estabelecem políticas de privacidade para provedores de serviços que possam ser compreensíveis e utilizáveis para os usuários, ao mesmo tempo em que as ferramentas de usuários, como os navegadores, tentam alertar seus usuários da forma mais simples possível dos riscos a privacidade [9].

2.4 Plataformas e Especificações de Gerenciamento de Identidades

Existem diversas especificações e plataformas de gerenciamento de identidade. As mais importantes são: OpenID, *Security Assertion Markup Language* (SAML) e InfoCard. Um dos principais pontos em comum desses protocolos é o *Single Sign-On*, onde o usuário precisa se autenticar em seu provedor de identidades uma única vez para ter acesso aos serviços. Além disso, o *Single Log-Out* (Encerramento Único de Sessões) permite a um usuário fechar todas as suas sessões abertas em provedores de serviços uma única vez.

2.4.1 *Security Assertion Markup Language*

O SAML define um padrão para troca segura de informações, criado pela OASIS. Define uma gramática XML para essa troca de dados. O SAML é bastante extensível e pode ser adotado por outros padrões de gerenciamento de identidade. Ele pode ser dividido em [10]:

- Asserções: documento XML que carrega informações sobre o usuário;
- Protocolo: define protocolos de pedido e resposta, codificados em XML;
- Transporte: detalha como os protocolos vão ser transportados pelas camadas inferiores, por exemplo: HTTP, HTTP sobre SSL, *Simple Object Access Protocol* (SOAP), etc;
- Perfis: Asserções, protocolos e *bindings* podem ser combinados para se descrever perfis diferentes. Por exemplo: SSO: define um mecanismo de autenticação mútua sem exigir alterações do navegador;

2.4.1.1 SAML e Segurança

O SAML pode usar protocolos diferentes dos definidos na especificação inicial. Um exemplo é usar a especificação WS-Security [11], que define mecanismos de segurança em mensagens SOAP [12]. Outra extensão possível para o SAML é usá-lo junto do XACML (*eXtensible Access Control Markup Language*), outra especificação da OASIS e possuí um perfil definido no próprio SAML.

2.4.2 InfoCard

InfoCard trata a troca de dados de usuários como troca de "cartões de informação". Esses cartões contêm uma série de afirmativas. Essas afirmativas podem ser de diferentes tipos: "Eu sou o usuário X", "Eu sou maior de idade", etc. Cada cartão é composto por uma série de afirmativas e são salvos dentro de arquivos XML.

Quando o usuário se conecta a um serviço, esse serviço envia a sua política e pede que o usuário escolha um determinado cartão de informação de acordo com ela. O usuário escolhe um determinado cartão, que é validado em seguida pelo provedor de identidades. O provedor de identidades retorna ao usuário um "cartão

seguro". Esse cartão pode ser um arquivo XML assinado e transmitido de forma segura, ou um certificado contendo os dados, por exemplo. Em seguida o usuário é redirecionado para o provedor de serviços e envia o cartão. O provedor de serviços verifica se é um cartão válido, caso afirmativo, permite que o usuário acesse o serviço desejado [13] [1].

2.4.2.1 InfoCard e Segurança

O protocolo permite que provedores de serviços enviem suas políticas de privacidade e que o usuário escolha o InfoCard que mais lhe convenha e esteja de acordo com essa política. Essas políticas podem ser expressas como previsto na especificação *WS-MetadataExchange* [14]. Além disso, a segurança utilizada nesse protocolo é próxima da prevista para o SAML, com mensagens SOAP seguras e uso de documentos XML criptografados.

2.4.3 OpenID

O OpenID é um conjunto de especificações cujo principal objetivo é definir um protocolo de autenticação entre os provedores de serviços, o provedor de identidades e usuários. A principal vantagem do OpenID é que seu protocolo de autenticação é simples, pois a sua adoção exige poucas alterações do provedor de serviços e nenhuma por parte do usuário. Essa característica foi determinante para a adoção do OpenID em alguns dos principais provedores de serviços da Internet, como por exemplo o Yahoo! e o Google. O OpenID vai ser estudado com mais detalhes no Capítulo 3.

Capítulo 3

O Protocolo OpenID

De acordo com [15], em 2009, o OpenID contava com mais de 500 milhões de usuários cadastrados e mais de 48 mil Provedores de Serviços o adotaram. A adoção do OpenID é simples, tanto para um usuário, exigindo apenas que este se cadastre em um Provedor de Identidades, quanto para um Provedor de Serviços.

O OpenID define uma série de especificações. A mais importante é o protocolo de autenticação, que atualmente se encontra na versão 2.0. Esse protocolo define mensagens HTTP trocadas entre o usuário, provedor de identidades (IdP) e provedor de serviços (ou *Relying Party* – RP), como são chamados na especificação do protocolo e podem ser vistos na Figura 3.1. As mensagens têm como objetivo autenticar o usuário junto ao IdP e em seguida redirecioná-lo ao serviço que este deseja acessar [16].

O OpenID permite que um usuário tenha posse de um Identificador Uniforme de Recursos (*Uniform Resource Identifier* - URI), chamado na terminologia de "identificador". Uma URI é um tipo especial de Localizador-Padrão de Recursos (*Uniform Resource Locator* - URL). Através da URI, um Provedor de Serviços pode identificar o usuário que deseja se autenticar e também qual o provedor de identidades (IdP) ele deve procurar para realizar a autenticação [17] [16].

3.1 Terminologia do Protocolo

- Usuário Final (*End User*): é o usuário que deseja acessar um determinado serviço;

- Consumidor ou Provedor de Serviços (SP) ou *Relying Party* (RP): é o Provedor de Serviços que o Usuário Final quer ter acesso. É chamado de Consumidor,

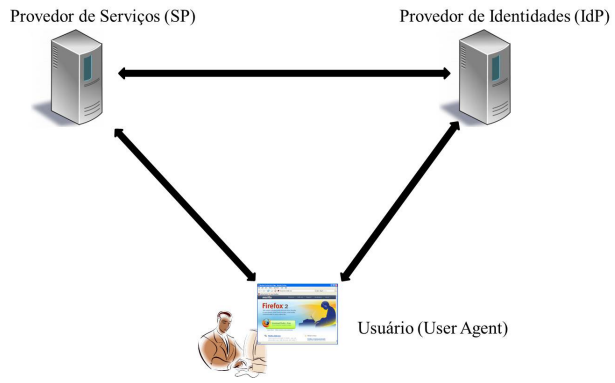

Figura 3.1: As três entidades participantes: usuário, provedor de identidades(IdP) e provedor de serviços(SP).

pois este "consome" as credenciais fornecidas pelo IdP;

- Identificador: URI ou XRI (*eXtensible Resource Identifier* - Identificador de Recursos eXtensível) que identifica o Usuário Final;

- Identificador Declarado (*Claimed Identifier*): identificador que um Usuário Final declarou ser, o objetivo do OpenID é confirmar se essa declaração é válida ou não. Esse Identificador Declarado deve ser utilizado pelo SP quando quiser guardar dados sobre um determinado usuário;

- Provedor de Identidades (IdP) ou Provedor de Identidades OpenID(*OpenID Provider* - OP): é o provedor junto ao qual o Usuário Final deve se autenticar para ter acesso ao serviço desejado na SP. Esse Provedor contém as credenciais do usuário e as fornece para o SP assim que o procedimento de autenticação é bem-sucedido;

- Agente do Usuário (*User Agent*): agente pelo qual o Usuário Final interage com o IdP e o SP. No caso, é um navegador.

A sigla SP foi adotada para provedor de serviços, apesar de que, RP também pode ser utilizado para definir provedor de serviços e é a sigla utilizada nas especificações do OpenID. Enquanto que OP vai ser utilizado para definir um provedor de identidades capaz de autenticar um usuário utilizando o protocolo de autenticação previsto no OpenID.

3.2 As Especificações do OpenID

O OpenID é composto por algumas especificações que visam definir pontos importantes no gerenciamento de identidades. Essas especificações descrevem protocolos e padrões para mensagens HTTP. Os principais focos dessas especificações são: autenticação de usuários, segurança na troca de mensagens entre as entidades, como transmitir informações dos usuários entre os provedores ou quais dados o provedor de identidades OpenID (OP) deve guardar sobre seus usuários. As especificações finalizadas são:

- Autenticação (*OpenID Authentication 2.0* [16]): apesar dessa especificação não se preocupar com o método de autenticação que vai ser utilizado entre o provedor de identidades OpenID (OP) e o usuário, ela define um protocolo de mensagens HTTP que devem ser trocadas para que um provedor de serviços (SP) possa verificar se um usuário é quem ele alega ser junto a um provedor de identidades OpenID (OP). Além disso, é definido também um padrão para essas mensagens;

- Troca de Atributos (*OpenID Attribute Exchange Extension 1.0* [18]): define um padrão para transmitir dados sobre o usuário, do provedor de identidades OpenID (OP) para o provedor de serviços (SP). Essas mensagens são transmitidas durante a autenticação, como definida em [16]. Os dados são passados através de campos das mensagens HTTP;

- Dados de Registro (*OpenID Simple Registration Extension 1.0* [19]): essa especificação estabelece alguns dados que o provedor de identidades OpenID (OP) deve obter do usuário quando este for se registrar. Esses dados são o mínimo necessário para se montar um perfil;

- Política de Autenticação para Provedores (*OpenID Provider Authentication Policy Extension 1.0* [20]): essa especificação analisa alguns dos procedimentos mais conhecidos de autenticação entre um provedor e seus usuários, classificando as formas de autenticação em níveis de segurança. O protocolo de autenticação OpenID [16] permite que o provedor de serviços (SP) defina o nível de segurança do procedimento de autenticação entre o provedor de iden-

tidades OpenID (OP) e o usuário devem utilizar. Quando o OP redireciona o usuário de volta para o SP, o OP deve informar qual o procedimento de autenticação foi utilizado. Essa comunicação entre os dois provedores é feita com campos adicionados às mensagens trocadas durante o protocolo.

A primeira especificação [16] define a comunicação entre as três entidades para que o usuário possa ser autenticado e acessar um determinado serviço. Em [18] e [19], definem-se os dados dos usuários necessários para construir suas identidades. Essas duas especificações focam como o SP pode requisitar informações sobre um determinado usuário durante a autenticação, como transferi-las do OP para o SP e quais devem ser os dados mínimos para que o OP possa construir um perfil do usuário. A última especificação [20] foca em segurança no procedimento de autenticação.

Esse projeto foca no protocolo de autenticação OpenID. Assim, o sistema de gerenciamento de identidades implementado é capaz de tratar as requisições de provedores de serviços (SPs) de acordo com esse protocolo. Além disso, os usuários devem fornecer os dados definidos na especificação de registro [19] para se cadastrar no sistema. A especificação sobre segurança na autenticação [20] foi utilizada para implementar um sistema que possa ser o mais seguro possível. Portanto sempre que este projeto de fim de curso se referir a "protocolo OpenID", está se referindo ao protocolo de trocas de mensagens para autenticar um usuário, como definido em [16].

3.3 O Procedimento de Autenticação

O protocolo OpenID estabelece um padrão para a sequência de mensagens HTTP trocadas entre as entidades envolvidas, definindo quais os passos necessários para a autenticação e também como os dados devem ser transferidos de forma segura. O protocolo OpenID encontra-se atualmente em sua segunda versão.

Há duas formas de comunicação entre os provedores no protocolo [16]:

- Comunicação Direta: duas entidades se comunicam usando o protocolo HTTP. Essa comunicação é geralmente utilizada para estabelecer associações ou verificações. Esses dois pontos vão ser explicados mais tarde;

- Comunicação Indireta: duas entidades conversam através de uma terceira. Essa é a comunicação mais comum, o agente final de um usuário geralmente executa o papel de entidade intermediária. A comunicação é feita através de HTTP *Redirects* ou redirecionamento via HTML .

  O protocolo pode ser resumido da seguinte forma (Figura 3.2):

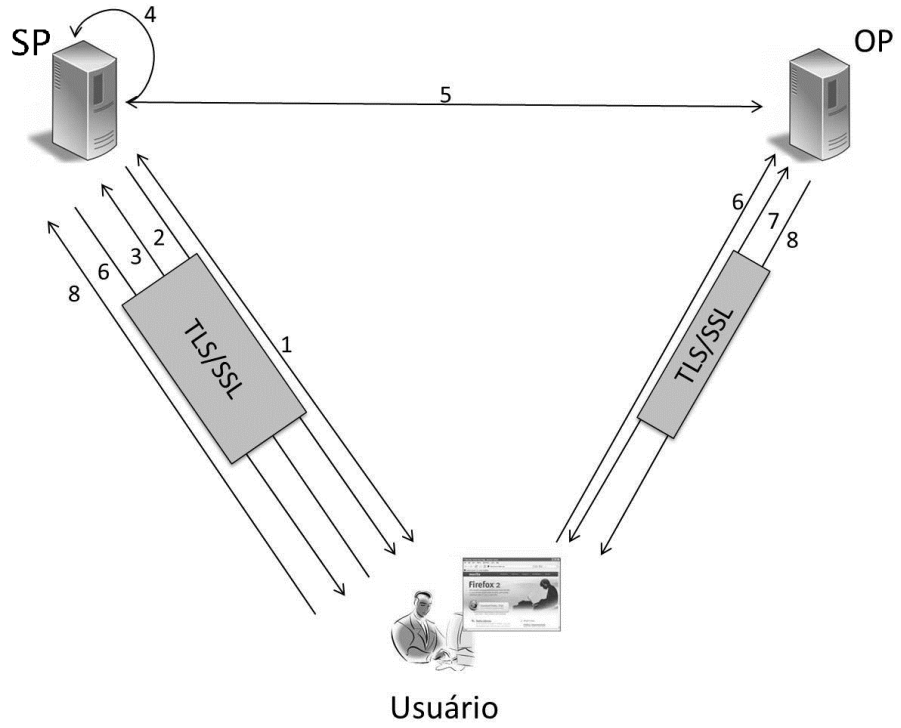

Figura 3.2: Sequência de mensagens trocadas entre as entidades durante a execução do protocolo de autenticação do OpenID.

1. o usuário acessa o sítio do provedor de serviços (SP);
2. o SP envia-lhe a página de autenticação;
3. o usuário insere o seu identificador OpenID;
4. o SP normaliza o identificador;
5. o SP executa o "Descobrimento" e encontra no identificador o provedor de identidades OpenID correspondente. O SP e o OP realizam uma associação (opcional), isto é, estabelecem uma chave comum que vai ser usada pelo restante do processo. Essa operação pode ser efetuada uma única vez, e os dois provedores memorizam os dados;

6. o usuário é redirecionado para a página do provedor de identidades OpenID (OP);

7. o usuário se autentica junto ao OP;

8. o OP redireciona o usuário para a sua página principal no serviço. Caso a autenticação tenha sido bem-sucedida, o SP recebe então uma mensagem denominada "asserção positiva", caso contrário, este receberá uma "asserção negativa".

Além dessas mensagens, o SP pode fazer um pedido de verificação diretamente para o OP (ou provedor de identidades OpenID). Esse pedido é feito após receber uma asserção (passo 8) do OP e tem por objetivo de verificar diretamente com o provedor OpenID se uma asserção de fato está correta ou não. A principal utilidade dessa mensagem é manter a compatibilidade entre as versões do OpenID (no caso, as especificações 1.1 e 2.0).

3.3.1 Descobrimento

O protocolo se inicia com o usuário informando o seu identificador. O Provedor de Serviços executa o Descobrimento ("Discovery"). Nessa operação, o provedor de serviços (SP) tenta descobrir informações necessárias para o procedimento de autenticação, contatando o OP, que lhe envia um arquivo XRDS (*eXtensible Resource Descriptor Sequence*) ou HTML descrevendo que tipos de serviços de autenticação ele tem disponíveis. Existem três formas de Descobrimento:

- Identificação XRI: esse identificador é resolvido em um arquivo XRDS de acordo com [21];

- Identificação através de uma URI: antes, o protocolo Yadis é tentado [22], se bem sucedido, um arquivo XRDS é recebido. Em último caso, o OP deve retornar um arquivo HTML contendo um cabeçalho(*header*) com os dados do protocolo OpenID.

Tanto o arquivo HTML quanto o XRDS retornados durante o descobrimento descrevem quais especificações do OpenID o provedor de identidades suporta, por exemplo, a troca de atributos prevista em [18].

3.3.2 Associação

Uma vez que o descobrimento retornou um documento válido, o provedor de serviços (SP) tenta se associar com o provedor OpenID (OP). O objetivo da associação é estabelecer um segredo comum que apenas esses dois provedores conhecem. Esse segredo vai ser utilizado para assinar as mensagens seguintes. A assinatura tem por objetivo principal de identificar para um dos provedores que a mensagem foi criada pelo outro provedor, além de garantir a integridade de seu conteúdo.

A assinatura da mensagem é feita através de um *Keyed-Hashing for Message Authentication Code* (HMAC). A especificação do OpenID define três tipos de associação [16]:

- HMAC-SHA1: define o uso do *Secure Hash Algorithm* 1(SHA1) como HMAC. O resultado desse algoritmo é um número de 160 bits;
- HMAC-SHA256: define o uso do SHA256 como HMAC. O resultado desse algoritmo é um número de 256 bits;
- DH-SHA1/DH-SHA256: define que haverá uma troca de chaves Diffie-Hellman para se estabelecer o segredo comum entre as partes. Além disso, define também qual algoritmo de *hash* vai ser utilizado;
- sem segredo: quando uma associação entre os provedores não é possível, o provedor de identidades cria um segredo que apenas ele conhece caso o usuário seja corretamente autenticado. Esse segredo é utilizado para assinar a mensagem e o SP deve efetuar um pedido de verificação de assinatura ao receber a mensagem de asserção positiva (passo 5 na Figura 3.2). O OP responde a essa mensagem com uma confirmação ou negação.

Para garantir essa segurança, deve-se garantir que apenas essas duas entidades sejam capazes de realizar a associação para se chegar ao segredo em comum. Em [16], propõe-se utilizar o protocolo Diffie-Hellman onde a chave estabelecida ao final vai ser o segredo comum. Outra proposta na especificação é utilizar um canal de transportes seguro, como o TLS ou SSL.

O aconselhável é utilizar a troca de chaves Diffie-Hellman através de um canal seguro, pois em canais seguros acontecem a troca de certificados válidos, o

que garante que as duas entidades que estão dialogando são de fato o provedor de serviços (SP) e o provedor de identidades OpenID(OP). Caso contrário, é possível efetuar ataques de negação de serviços que provoquem solicitações que possam exaurir recursos de processamento, como por exemplo, o atacante simula um OP ou um provedor de serviços (SP) e obriga o outro provedor a executar o algoritmo de Diffie-Hellman. Outro ataque possível é "phishing", onde um atacante simula ser o OP e consegue enganar o provedor de serviços (SP) e estabelecer um segredo comum.

Uma associação pode durar mais do que a sessão de usuário, uma vez que o segredo comum foi estabelecido, basta memorizá-lo. Essa política fica a cargo o OP e do SP de gerenciar. De qualquer forma, esse segredo deve ser refeito de tempos em tempos para garantir a segurança do processo.

3.3.2.1 Assinatura de Mensagens

Certas mensagens devem ser assinadas pelos provedores para garantir a integridade e autenticidade de seu conteúdo. Essa assinatura é feita através de um algoritmo de *hash* do valor de certos campos da mensagem HTTP juntamente com um segredo. Esse segredo é definido durante a associação e ambos os provedores conhecem.

A assinatura de mensagens por um *hash* dos seus campos e um segredo garante, primeiro, que apenas um provedor autêntico pode ter enviado a mensagem, uma vez que apenas os provedores de identidade OpenID (OP) e de serviços (SP) conhecem esse segredo. Segundo, os valores de certos campos importantes para essa mensagem são utilizadas na assinatura, o que garante a integridade dessas mensagens. Ou seja, nenhum atacante alterou o valor desses campos.

3.3.2.2 Troca de Chaves Diffie-Hellman na Associação

O objetivo da troca de chaves Diffie-Hellman é ao final do procedimento, estabelecer uma chave secreta comum entre as duas entidades, no caso o provedor de serviços (SP) e o provedor de identidades OpenID (OP). O algoritmo pode ser resumido nos seguintes passos:

1. ambas as partes estabelecem um número primo p e um número g em comum;

2. SP escolhe aleatoriamente um número a e apenas ele vai conhecer durante

todo o processo de troca. Esse número deve ser um inteiro positivo e menor ou igual a $p - 2$;

3. OP também escolhe aleatoriamente um número b que apenas ele conhece e que seja menor ou igual a $p - 2$;

4. SP calcula o valor de A

$$A = g^a \pmod{p}; \tag{3.1}$$

5. OP calcula o valor de B

$$B = g^b \pmod{p}; \tag{3.2}$$

6. As partes trocam os números A e B;

7. os provedores SP e OP calculam o valor do segredo comum (sg), de acordo com

$$sg = A^b \pmod{p} = B^a \pmod{p}. \tag{3.3}$$

.

Para ambas as entidades, descobrir o valor do segredo comum sg é fácil, uma vez que basta elevar o valor recebido pelo número secreto (a ou b). A segurança desse algoritmo de troca de chaves é que um atacante, mesmo conhecendo os valores de g e de p, tenha que executar uma série de logaritmos e exponenciais para encontrar os valores corretos de a e b e chegar até o segredo comum [23].

No OpenID essa troca de chaves é feita através de campos nas mensagens HTTP [16]. O provedor de serviços (SP) tenha o número secreto a e o provedor de identidades OpenID (OP) tenha o número secreto b. A associação começa com o SP enviando uma mensagem HTTP ao OP. Essa mensagem contém os seguintes campos:

- openid.dh_modulus: valor de p;
- openid.dh_gen: valor de g;
- openid.dh_consumer_public: valor de A (como calculado pela Equação 3.1).

O OP escolhe um número, conhecido como *mac_key*, que vai ser o segredo comum entre os provedores. Em seguida, ele responde ao SP com uma mensagem HTTP contendo os seguintes campos, onde a função Hash(.) significa um *hash* do tipo SHA1 ou SHA256:

- dh_server_public: valor de B (como calculado pela Equação 3.2);
- enc_mac_key: O segredo comum (chamado de *mac_key* no protocolo OpenID), é enviado nesse campo, de acordo com a Equação 3.4,

$$enc\_mac\_key = Hash(A^b \pmod p) \oplus mac\_key. \tag{3.4}$$

O segredo comum entre as duas entidades é o *mac_key* da mensagem de resposta. Esse número é escolhido pelo provedor de identidades OpenID (OP) e enviado pelo campo *enc_mac_key* nessa mensagem de resposta. Para que o provedor de serviços (SP) encontre esse valor, deve-se utilizar o valor de B (recebido no campo *dh_server_public*) e calcular o valor do segredo comum de acordo com

$$mac\_key = enc\_mac\_key \oplus Hash(B^a \pmod p). \tag{3.5}$$

Capítulo 4

Cartões Inteligentes e Autenticação Segura

Autenticação é o processo de verificar a identidade de alguém ou algo [24]. No caso deste trabalho, o interesse é autenticar um determinado usuário para serviços de Internet. Há várias formas de autenticação em redes de computadores e podem ser divididas em três tipos:

1. o que o usuário sabe;
2. o que o usuário tem;
3. quem o usuário é.

O Tipo 1 é a forma mais popular de autenticação na Internet: uso de senhas que apenas um usuário sabe. O Tipo 2 é, por exemplo, o uso de cartões eletrônicos de banco, onde um usuário prova a sua identidade quando insere seu cartão e digita uma senha simples em um caixa eletrônico. O Tipo 3 é o uso de impressões digitais, reconhecimento íris ou qualquer outro recurso biométrico para provar a sua identidade.

A senha, apesar de ser a forma de autenticação mais popular, também é a forma mais insegura dos tipos acima. Os problemas com o uso de senhas para autenticação são:

- alguém pode interceptar a senha quando ela é transmitida ou espionar o computador do usuário, usando *keyloggers* que memorizem tudo o que o usuário digita;
- a senha pode ser fraca e um intruso pode adivinhá-la;

- suscetível a "phishing": um atacante pode redirecionar o usuário para um site falso para roubar a senha, onde este usuário inadvertidamente entra a sua senha;

- obriga o usuário a memorizar a senha. Caso ele esqueça, deve passar por um procedimento para restaurar a sua identidade.

4.1 Segurança na Autenticação

Usar senhas como uma forma de se autenticar é um processo falho e cada vez mais vulnerável. A senha é do tipo "o que o usuário sabe" e para aumentar a segurança desse sistema, pode-se exigir que o usuário apresente um determinado objeto (*token*) no momento da autenticação, caracterizando também uma autenticação pelo "que o usuário possui".

O usuário deve apresentar um cartão e uma senha específicos para poder ter acesso às suas informações. Esses cartões pertencem a uma classe conhecida como cartões inteligentes (*smart cards*).

Usar a autenticação do tipo "o que o usuário possui", apresenta algumas desvantagens [24]:

- O usuário precisa sempre ter seu cartão em mãos para ser autenticado;

- Usar um cartão nem sempre abole o uso de senhas. A autenticação do tipo "o que o usuário possui" normalmente é acompanhada de outro tipo de autenticação, como por exemplo uma senha de acesso, ou "o que o usuário sabe". Assim, em caso de roubo do cartão, o malfeitor não pode se autenticar como sendo o usuário;

- quase sempre exige o uso de equipamentos especiais, como leitores de cartões.

4.2 Cartões Inteligentes - *Smart Cards*

Os primeiros cartões usados para autenticação do tipo "o que o usuário possui" eram feitos de plástico e os dados elementares, como por exemplo o nome da empresa que fez o cartão, o nome do usuário e o código desse cartão eram impressos em alto relevo. Havia também um espaço onde o usuário deveria colocar a sua

assinatura. O procedimento de autenticação era feito através do reconhecimento da assinatura por uma pessoa, que comparava a assinatura do usuário com a assinatura no cartão.

Esse sistema era falho e uma série de fraudes obrigaram as empresas a procurarem formas mais seguras de identificação, surgindo assim as tarjas magnéticas. Nessa nova arquitetura, o usuário insere uma senha e esses dados podem ser comparados com o conteúdo nas tarjas. A autenticação agora requer um dispositivo de leitura da tarja magnética, o que dificulta e torna mais dispendiosa a falsificação. O grande problema das tarjas magnéticas é que elas podem ser lidas e manipuladas por qualquer pessoa que tenha acesso aos equipamentos de leitura e escrita de tarjas magnéticas. Os cartões iniciais não possuíam, portanto, inteligência.

A evolução foi o cartão inteligente (*smart cards*). O cartão inteligente possui um processador embarcado inviolável, denominado microcontrolador seguro. Os cartões inteligentes (*smart cards*) começaram a ser fabricados na década de 80, após os grandes avanços na área de microeletrônica. Inicialmente, eram cartões com memória apenas, onde o usuário inseria sua senha e caso esteja correta, as informações contidas na memória são liberadas [25].

4.2.1 Tipos de Cartões Inteligentes

Os cartões inteligentes podem ser divididos em categorias, primeiro pela forma como são lidos, se há contato entre a leitora (*contact*) e os cartões ou não (*contactless*). A segunda categoria é:

- Cartões de Memória(*Memory Cards*): esses cartões contêm uma EEPROM, uma ROM e uma pequena unidade lógica, responsável por dar acesso a certos setores da memória se o usuário fornecer a senha correta ou bloquear o cartão caso alguém tenha feito várias tentativas malsucedidas para acessá-lo;

- Cartões com Processadores (*Microprocessor Cards*): esses cartões contêm um sistema embarcado inteiro em seu interior. Nesse sistema, existem processadores, memória RAM e ROM e geralmente, um coprocessador para operações criptográficas. Esses cartões possuem sistemas operacionais e aplicações desenvolvidas.

Os cartões inteligentes (ou *smart cards*) com microprocessadores, também conhecidos como Cartões de Circuito Integrado (*Integrated Circuit Cards* – ICCs) possuem uma padronização internacional. O padrão é conhecido como ISO 7816, que define desde as características e dimensões do cartão até o protocolo de mensagens trocadas entre o cartão e o terminal [25].

4.2.2 A Comunicação com os Cartões Inteligentes

A ISO 7816 define vários protocolos de comunicação. Para esse trabalho, o mais importante é a troca de mensagens entre a aplicação instalada sobre o sistema operacional do cartão e o terminal. A ISO 7816-4 define os formatos das mensagens nesse protocolo, tanto para os tipos de comando enviados pelo terminal quanto de resposta que o cartão deve retornar. Essas mensagens são conhecidas como APDUs ("Application Protocol Data Unit").

A APDU deve ser dividida em dois tipos: as mensagens enviadas do terminal para o cartão (APDUs de Comando) e as respostas dadas pelo cartão (APDUs de Resposta). O primeiro tipo de mensagem tem o formato visto na Figura 4.1.

Campo Obrigatório				**Campo Opcional**		
CLA	INS	P1	P2	Lc	Dados	Le

Figura 4.1: APDU de comando enviada do terminal para o cartão.

- CLA – *Class Byte* (*Byte* de Classe) - define a categoria do APDU;
- INS – Instruction Byte (*Byte* de Instrução) - especifica a instrução de comando;
- P1 – *Parameter Byte 1* (*Byte* de Parâmetro) - é o primeiro dos dois *bytes* de parâmetros para a instrução especificada;
- P2 – *Parameter Byte 2* (*Byte* de Parâmetro) - é o segundo byte de parâmetros para a instrução especificada;
- Lc – *Length Command Byte* - define a quantidade de *bytes* de dados que vão ser enviados nessa APDU;
- Dados – Varia de um a 256 *bytes* de dados enviados para o cartão;

- Le – *Length Expected* - define a quantidade de *bytes* de dados esperado na resposta do cartão.

A Figura 4.2 mostra o formato da mensagem APDU de resposta que o cartão inteligente envia para o terminal.

Campo Opcional	Campo Obrigatório	
Dados	SW1	SW2

Figura 4.2: APDU de resposta enviada pelo cartão para o terminal

- Dados – Esse campo tem o tamanho que varia de um a 256 *bytes* e corresponde aos dados enviados do cartão para o terminal. Não obrigatoriamente precisa ter o mesmo tamanho que o definido no byte "Le" de comando;
- SW1 – *Status Word - byte 1*, um dos dois *bytes* que retorna o status do funcionamento do cartão. Caso a palavra seja 0x9000, o processo terminou normalmente;
- SW2 – *Status Word - byte 2*, o outro *byte* de status.

O cartão trabalha sempre em modo escravo. O programa no terminal envia uma APDU de comando para o cartão, e a aplicação dentro do cartão trata essa mensagem e envia uma resposta.

4.3 Cartões Java - *Java Cards*

Uma alternativa para facilitar o desenvolvimento de aplicações para cartões inteligentes é criar um padrão de programação que possa rodar em vários modelos diferentes de cartões. Uma plataforma capaz de tal versatilidade é a linguagem Java e as máquinas virtuais Java. Seguindo esse conceito, a especificação *Java Card* prevê um modelo de programação que procura ser independente do cartão e do sistema operacional. Assim, o sistema fica apenas dependente da versão Java usada.

Existem diversas versões dessa especificação. A especificação *Java Card 1.0* foi a primeira versão, elaborada pelo *Java Card Forum*, um consórcio de diversas empresas que visavam estabelecer a tecnologia de cartões inteligentes no mercado e

criar um padrão de programação. A versão 2.0 veio quando a *Sun Microsystems, Inc.* assumiu a responsabilidade de desenvolver a linguagem *Java Card*. A especificação 3.0 permite maior interatividade onde o cartão poderá agir como servidor e interpretando protocolos de rede como o HTTP e o HTTPS. Aplicações escritas para cartões java são chamadas de *Java Applets*. Nesse capítulo elas vão ser apelidadas de *applets*.

Da especificação 2.0 em diante, o funcionamento dos cartões java em três partes: primeiramente, o *Java Card Runtime Environment (JCRE)*, responsável por gerenciar recursos, execução de *applets*, receber e encaminhar APDUs, etc. Dentro do JCRE, existe a máquina virtual java (*Java Card Virtual Machine - JCVM*). Essa máquina virtual é responsável por interpretar as instruções de *applets*, gerenciar a memória, etc. A terceira parte é o *applet*. Essas aplicações são escritas pelos programadores e servem para personalizar cartões, permitindo que eles executem diversos tipos de tarefas diferentes utilizando-se as classes previstas na especificação. [26].

4.3.1 O Modelo de Cartão Selecionado

A especificação do OpenID conhecida como *OpenID Provider Authentication Policy Extension* [20] define políticas de autenticação e seus respectivos níveis de segurança, que podem ser utilizadas entre o provedor de identidades OpenID (OP) e o usuário. Essa especificação define quatro níveis de segurança para processos de autenticação, onde o quarto nível garante a segurança máxima possível, evitando todos os tipos de ataques discutidos nas especificações [16] [20]. O uso de apelido e senha é a segurança mais baixa, enquanto que o máximo de segurança é possível utilizando-se um cartão inteligente capaz de realizar operações de criptografia em seu interior. Esse cartão deve atender a norma FIPS140-2 [27], que prevê que o circuito eletrônico do cartão deve ser protegido com um revestimento. O revestimento deve ser capaz de evitar que um atacante possa medir sinais dentro do circuito, além dos de entrada e saída. Portanto, para que um malfeitor possa chegar às chaves guardadas dentro do cartão, ele deve remover o revestimento, o que denuncia que o cartão foi adulterado.

O modelo de cartão selecionado é o JCOP21, desenvolvido e inicialmente comercializado pela IBM. Atualmente ele é produzido pela NXP. Esse cartão usa

leitores com contato e contém uma máquina virtual para cartões java versão 2.1.1, sendo capaz de conter diversos *applets* dentro do mesmo cartão. A seleção de qual *applet* executar é feita através de um APDU, logo após conectar o cartão no leitor. O cartão permite executar diversos algoritmos de criptografia. O cartão atende à norma FIPS140-2 nível 3 física [27] [28], possuindo um microcontrolador de 8 bits com dois coprocessadores: um para chaves simétricas e outro para chaves assimétricas.

O Apêndice B entra em detalhes da aplicação desenvolvida para o JCOP21 para esse projeto, discutindo o funcionamento da memória nesses tipos de cartões, o ambiente de desenvolvimento empregado e as partes mais importantes do código dessa aplicação.

Capítulo 5

Gerenciador de Identidades OpenID com Cartões Inteligentes

O projeto estuda e implementa um sistema de gerenciamento de identidades baseado nas especificações do OpenID e utiliza cartões inteligentes (*smart cards*) na autenticação. Essa arquitetura possui uma segurança bem maior que o popular apelido e senha da Internet.

Nesse capítulo, o provedor de identidades (ou provedor OpenID) também vai ser chamado de forma indistinta de servidor OpenID.

5.1 Objetivos

O projeto tem por objetivo de criar uma forma mais segura de administrar a identidade digital. Primeiramente, protocolos de gerenciamento de identidade foram estudados e o OpenID foi escolhido por sua ampla aceitação e simplicidade. Ele possui fraquezas já conhecidas [20] [29]. A forma mais segura de se utilizar o OpenID [20] é uma autenticação mútua na qual o usuário possui um microcontrolador seguro capaz de executar operações criptográficas em seu interior e executar o protocolo HTTPS. Cartões inteligentes (*smart cards*) são capazes de executar esses algoritmos de criptografia, como o RSA. Portanto, esse projeto visa elaborar e implementar uma arquitetura de um gerenciador de identidades seguro usando OpenID com cartões inteligentes, como a arquitetura proposta (Figura 5.1). A autenticação é feita de forma mútua, tanto o provedor de identidades deve verificar se o usuário é quem afirma ser quanto o cartão deve verificar se o provedor de identidades é o correto.

5.2 Trabalhos Relacionados

Os trabalhos diretamente relacionados a esse tema abordam a utilização do OpenID juntamente com algum microcontrolador seguro. Os trabalhos relacionados focam tanto a parte de implementação da comunicação entre um microcontrolador e as entidades envolvidas do OpenID quanto possíveis aplicações comerciais dessa solução.

Uma das primeiras propostas de se utilizar o OpenID com microcontroladores foi feita por Jhrstad *et al.* [15] de autenticar, no OpenID, com algoritmos de criptografia previstos no GSM/UTMS para se autenticar em um provedor de identidades. Esses algoritmos são executados dentro de *SIM cards*. Os autores defendem que a arquitetura que eles propõem é melhor que a autenticação por SMS, na qual um usuário recebe uma mensagem de texto contendo uma OTP(*One Time Password*) em seu celular, que vai autenticá-lo em um determinado serviço. Jhrstad *et al.* propõem que a operadora atue como provedora de identidades e, através do protocolo EAP-SIM [30], os algoritmos de criptografia (A3/A8), já implementados dentro do próprio *SIM card*, possam ser utilizados em uma autenticação mútua. Quando um usuário quiser usar um serviço de um provedor de serviços a partir de um navegador, ele é redirecionado para um provedor de identidades OpenID mantido pela operadora e se autentica com ela. A comunicação entre o *SIM card* e esse provedor OpenID é feita através de um *Java Applet* do sítio. Desta forma, o usuário assim como o provedor de identidades OpenID da operadora são autenticados mutuamente, evitando assim *phishing*.

Leicher e Schmidt [31] estenderam a proposta anterior com cartões SIM capazes de agir como cartões inteligentes. O cartão agora recebe funcionalidades antes restritas ao provedor de identidades. Urien, Marie e Kiennert propõem em [32] o uso de cartões inteligentes capazes de executar o protocolo EAP-TLS [33]. O artigo propõe o gerenciamento de identidades assumindo que apenas os cartões inteligentes e os provedores são repositórios seguros de informações. A prova da identidade é, portanto, feita apenas por componentes seguros. Cartões capazes de abrir conexões SSL com servidores são utilizados para autenticação mútua. O cartão autentica o servidor e o servidor autentica o certificado X509 que está dentro do cartão. Além disso, os autores discutem o ciclo de vida de identidades. Um usuário primeiro se

registra no provedor de identidades, cadastrando os dados necessários para provar a sua identidade e deve encomendar um novo cartão caso o seu antigo tenha sido perdido. Usuários podem guardar parte de suas informações dentro do microcontrolador. A comunicação entre o cartão e o computador do usuário é feita através de um *proxy* [34]. A proposta do artigo é de utilizar cartões java capazes de tratar comunicações EAP-TLS. Quando um usuário quer se conectar através do cartão, ele fornece a URL de destino. De acordo com [34], a construção de um canal SSL entre o cartão e um servidor leva 4.2 segundos, para os cartões mais rápidos. A taxa máxima de tranferência entre ambos está abaixo de 2kbytes/s. Portanto, utilizar um cartão inteligente como intermediário de uma conexão na Internet é impraticável. Por esse motivo, o *proxy* estabelece uma conexão SSL, com a troca de certificados entre o cartão e o servidor e a formação de uma chave de sessão. Por motivos de desempenho, essa chave de sessão é exportada do cartão para o *proxy* e a conexão SSL continua. Um risco sério de segurança que precisa ser tratado é a integridade desse *proxy*. Uma vez que um usuário pode, por exemplo, ser levado para um sítio falso, por *phishing*, e baixar um *proxy* pirata que vai interceptar ou manipular o seu tráfego. Algumas soluções são: assinar esse *proxy* e verificar a assinatura ou baixá-lo de servidores confiáveis.

Leicher, Schmidt e Shah também propõem [35] uma arquitetura de um provedor OpenID capaz de autenticar usuários que possuam TPMs (*Trusted Platform Modules*) [36]. O objetivo é associar a identidade de um determinado usuário à plataforma que ele está usando, assim como permitir que o provedor de identidades possa checar o estado da plataforma, como por exemplo, saber se o navegador não foi alterado, o que poderia significar um ataque. Para se registrar em um provedor OpenID, o usuário gera um par de chaves RSA dentro do TPM, esse par é conhecido como *Attestation Identity Keys* (AIK) e assinado por uma autoridade certificadora confiável. A autenticação é feita por um desafio (*challenge*) do provedor de identidades OpenID. A TPM deve criptografar esse desafio com a sua AIK correta e reenviar para o provedor. Esse protocolo infelizmente não prevê uma autenticação mútua.

5.3 A Arquitetura Proposta

Para elaborar a arquitetura do sistema de gerenciamento de identidades proposto neste projeto de fim de curso, algumas escolhas foram feitas:

- os modelos de cartão devem ser cartões inteligentes capazes de executar operações criptográficas como o RSA no *hardware* e, portanto, foram escolhidos os cartões java;
- o estabelecimento de uma comunicação direta entre o cartão e o servidor OpenID deve ser especificado. Para facilitar a aceitação do sistema por parte dos usuários, utilizou-se *Java Applets* nos navegadores, pois assim a comunicação deveria ser feita através de elementos do próprio navegador e exigiria no máximo que o usuário atualize a versão de seu navegador;
- a leitura e o processamento das mensagens HTTP pelo cartão e conversão das respostas do cartão para mensagens no formato HTTP são executados no *Java Applet* escrito para esse projeto. Esse *applet* tem duas tarefas. Primeiro, exibe o andamento da autenticação para o usuário e segundo, recebe mensagens HTTP e as transfere para o cartão através de APDUs ou converte uma sequência de APDUs em uma mensagem HTTP.

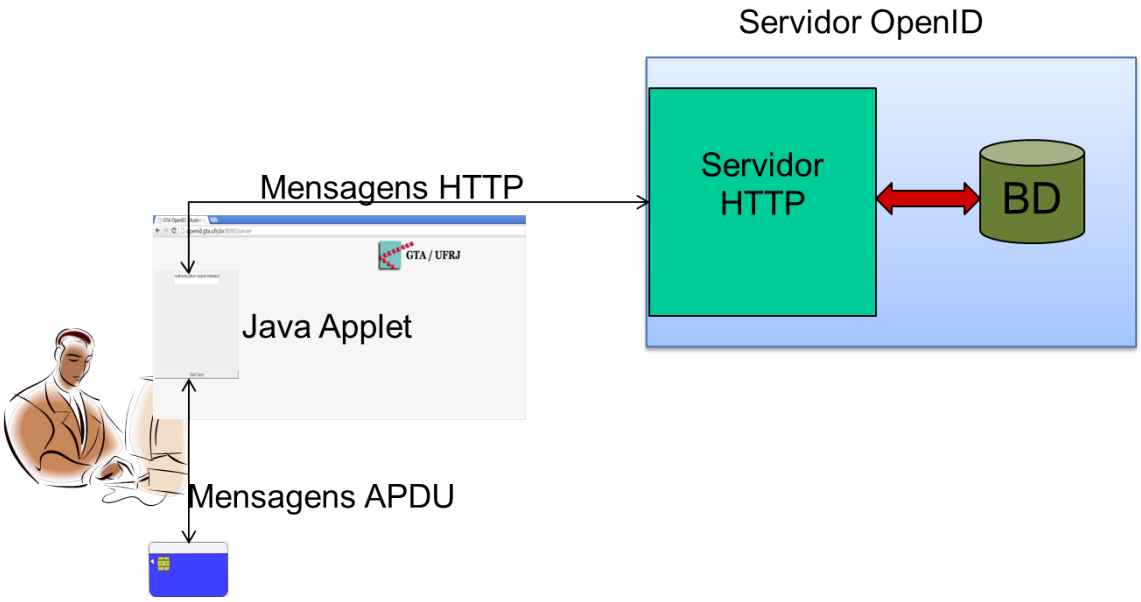

Figura 5.1: A Arquitetura do Sistema de Gerenciamento de Identidades.

A arquitetura é mostrada na Figura 5.1. Os principais blocos são:

- o servidor de identidades OpenID que é dividido em dois sub-blocos, o banco de dados, que grava os dados dos usuários e de seus cartões, e o servidor HTTP, que responde às mensagens;

- o agente do usuário (*User Agent*) que é o recurso que o usuário utiliza para se conectar ao servidor OpenID. No caso, qualquer navegador capaz de executar *Java Applets* versão 1.6 ou superiores. A escolha de se manter o navegador garante mais facilidades para os usuários, pois em nenhum momento o usuário precisa realmente intervir ou executar qualquer outro programa em seu computador;

- os cartões inteligentes que são entregues ao usuário com a aplicação capaz de se comunicar com o servidor já instalada. Essa aplicação deve executar dois protocolos, que vão ser discutidos mais adiante.

5.3.1 Ciclo de Vida da Identidade

O gerenciador de identidades é responsável por manter o ciclo de vida de uma identidade e dos cartões. Ou seja, ele deve permitir cadastro de novos usuários, modificação de dados desses usuários, fechamento de contas, registro e exclusão de cartões e a autenticação desses usuários.

Na Figura 5.2 pode-se ver o ciclo de vida de uma identidade e de seus cartões. O primeiro passo é o cadastro de dados no sítio. Uma extensão da especificação do OpenID define parâmetros básicos a se cadastrar [19]. Em seguida, os administradores do servidor OpenID devem emitir um cartão para esse novo usuário. Primeiro instala-se a aplicação no cartão e em seguida, registra-se o cartão no servidor. Esse registro pode ser feito localmente pelos administradores ou remotamente pelo próprio usuário. O protocolo de registro do cartão vai ser explicado mais tarde em detalhes. O cartão deve ser periodicamente revogado, tal como acontece com os certificados na Internet, para garantir a segurança do sistema, enquanto um novo cartão deve ser emitido para esse usuário. Ou seja, um único usuário pode ter vários cartões durante o ciclo de vida de sua identidade. Por último, o usuário pode pedir o fechamento de sua conta, o que leva à revogação de seu cartão atual.

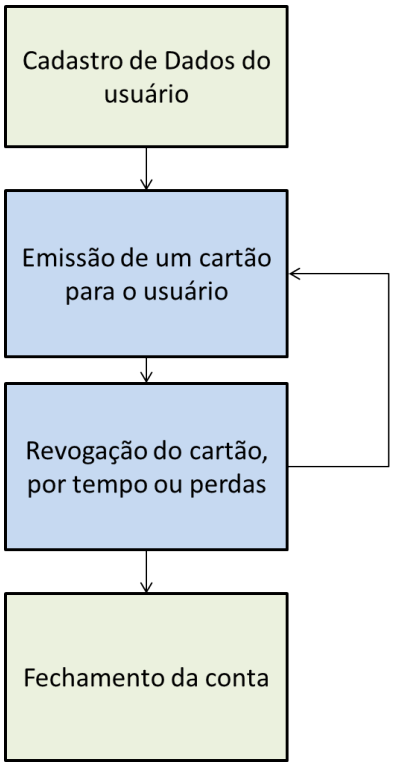

Figura 5.2: O Ciclo de Vida de uma Identidade do servidor.

5.3.2 Os Protocolos de Comunicação

Dois protocolos foram elaborados para o gerenciador de identidades proposto. O primeiro é o protocolo de registro, que executado uma única vez, logo depois de instalar a aplicação no cartão. Esse protocolo tem como objetivo cadastrar dados sobre o cartão no servidor OpenID. Essas informações são utilizadas durante a autenticação do usuário. O segundo protocolo define a autenticação e estabelece mensagens a se trocar entre o cartão e o servidor de forma a que ambos possam se autenticar. Ambos os protocolos utilizam criptografia assimétrica.

5.3.2.1 Protocolo de Registro

A Figura 5.3 resume o protocolo de registro. Ele deve ser executado logo após a instalação do aplicativo no cartão e tem por objetivo registrar as informações necessárias tanto no cartão quanto no servidor OpenID. Durante a instalação do aplicativo, o cartão deve gerar o par de chaves que lhe corresponderá.

O primeiro passo é identificar qual usuário e qual modelo de cartão se quer registrar no servidor. Todo o registro é feito via navegador. Uma página *HTML* contendo o *Java Applet* de registro se abre no navegador. Esse *applet* abre uma

conexão entre o cartão e o servidor. Ambos trocam informações necessárias para a autenticação:

- Chave Pública do Servidor: para o caso do algoritmo RSA, a chave pública é composta por expoente e módulo. Ambos são gravados dentro do cartão;
- Chave Pública do Cartão: mesmo caso da chave pública do servidor;
- número de cadastro do usuário e do cartão: cada usuário tem um número de cadastro, assim como seus cartões. Essa arquitetura não obriga que um usuário tenha apenas um cartão por vez e, portanto, não adianta que o cartão saiba apenas o número de cadastro do usuário, ele também deve ter um número próprio;
- dados do cartão: o cartão transmite o seu *Answer to Reset*, que possui dados sobre quem fabricou esse cartão, seu modelo, e outros dados. Essa informação fica guardada no servidor.

Na Figura 5.3 pode-se ver o algoritmo de registro implementado para esse servidor. O protocolo de registro pode ser feito pela Internet ou, por questões de

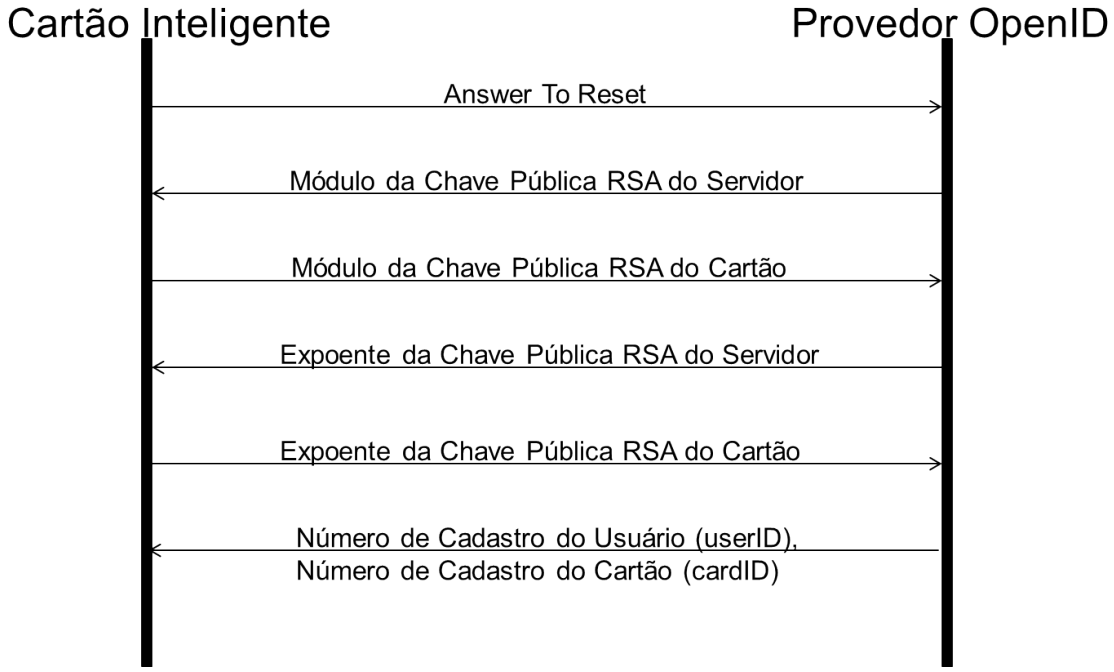

Figura 5.3: O Protocolo de Registro de Cartões.

segurança, o registro de cartões pode ser limitado a certos computadores. Essa última opção evita dois problemas:

1. o ataque de negação de serviços: como o servidor não sabe se está falando com um cartão de verdade ou com um programa simulando um cartão e como um usuário pode ter vários cartões, um atacante pode simular cartões para efetuar uma negação de serviços, que obriga o servidor a executar operações de busca e inserção no banco de dados. O custo para o atacante é de escolher números a esmo como expoente e módulo da sua "chave pública";

2. o ataque de *phishing*: um atacante pode se interpor entre o servidor OpenID e o usuário, quando ele for se registrar, levando o usuário a se registrar em um servidor pirata e colocando em sério risco a sua segurança.

O interesse de se liberar o registro de cartões remotamente é de reduzir o trabalho dos administradores do servidor e de outro lado abrir a possibilidade de novas propostas como a proposta feita por Akhram, Markantonakis e Mayes em [37]. Nesse caso, para se evitar ataques de negação de serviço, o servidor deve criar limitantes de quantos cartões cada usuário pode ter cadastrado ao mesmo tempo. O *phishing* pode ser evitado usando-se um canal seguro como o SSL, que exige que o servidor apresente um certificado assinado por uma autoridade de confiança.

5.3.2.2 Protocolo de Autenticação

O protocolo de autenticação define as mensagens trocadas entre o cartão e o servidor para verificar a identidade de seu usuário. A autenticação é mútua, ou seja, o servidor verifica que o usuário é quem diz ser e o cartão também verifica se o servidor é o correto. Para que o procedimento seja realmente seguro, o cartão deve ser capaz de atender as especificação definida em [20].

A Figura 5.4 resume o funcionamento do protocolo. Sua segurança vem do uso de um algoritmo de criptografia assimétrica, no caso desta implementação, o RSA. Para garantir a segurança, cada uma das partes conhece e utiliza a chave pública da outra para criptografar. Os passos são:

1. o cartão envia o número de cadastro do usuário e do próprio cartão, juntamente com um número gerado aleatoriamente que é usado como o desafio da autenticação para o servidor OpenID;

2. o servidor responde, enviando um número aleatório gerado que também é usado como desafio para a autenticação no sentido inverso;

3. o cartão decriptografa o número aleatório de desafio recebido e reenvia, provando conhecer a chave privada par à chave pública que o servidor conhece e com a qual a mensagem foi criptografada. Isto corresponde à autenticação do cartão pelo servidor;

4. o servidor responde com o número aleatório que este tinha recebido, provando possuir a chave privada do servidor. Isto corresponde à autenticação do servidor pelo cartão;

5. o cartão responde se a chave é a certa;

6. o servidor envia a URL para o qual o usuário deve ser redirecionado (caso esteja autenticando o usuário para um provedor de serviços) ou redireciona o usuário para a sua página de dados pessoais.

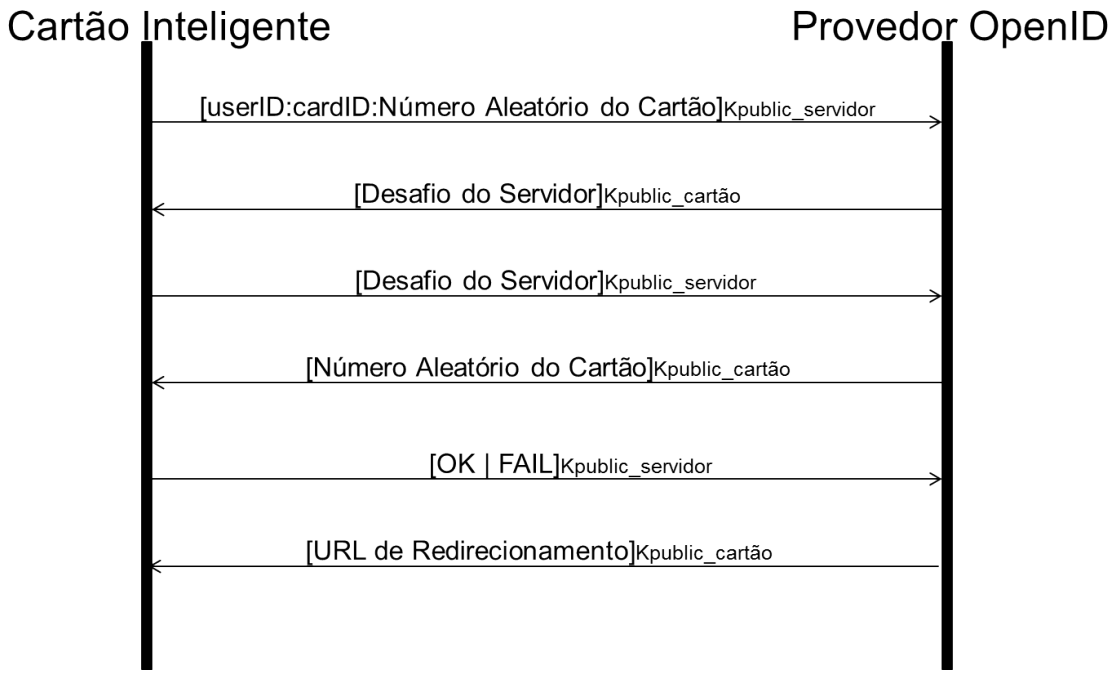

Figura 5.4: O Protocolo de Autenticação de Cartões.

Toda a segurança desse sistema vem do fato que apenas o cartão conhece a chave privada par à chave pública que o servidor conhece e que apenas o servidor conhece a sua própria chave privada. A etapa de registro serve para que o cartão

possa informar ao servidor a sua chave pública, assim como guardar a chave pública desse servidor.

5.4 A Implementação

A eficiência da arquitetura proposta foi testada usando-se um servidor web *Apache Tomcat 7*, no qual o provedor de identidades OpenID foi desenvolvido como um *Servlet* em Java. O banco de dados escolhido é o MySQL Server versão 5.5 e a comunicação entre o banco de dados e o *servlet* é através do uso da biblioteca MySQL-Connector/J.

Os cartões utilizados nessa implementação são modelos JCOP21 da NXP. Esses cartões possuem a JavaCard API 2.1.1. O servidor foi desenvolvido em Java utilizando a biblioteca openid4java [38]. O diagrama das principais classes do projeto pode ser visto no Apêndice A.

Nas próximas seções a implementação é descrita, dividida de acordo com o padrão de design MVC (*Model-View-Controller*). Neste padrão, o "model" (modelo) se preocupa em definir como os dados foram organizados, "view" (interface), na forma em que são exibidos e "controller" (controle), em como o usuário pode interagir com esse sistema.

5.4.1 Modelo

Os dados mais importantes para o servidor OpenID são os dados dos usuários e os dados necessários para autenticá-los. Todas essas informações são guardadas dentro de um banco de dados MySQL e a interação entre ambos é feita pela biblioteca Connector/J, para Java.

As interações com o banco de dados são divididas entre a classe "sqlConnector" e as classes que administram as informações (exemplo: "user"). A "sqlConnector" encapsula as chamadas ao Connector/J e é responsável por inicializar ou fechar as conexões com o MySQL, assim como organizar os dados em formatos adequados (por exemplo: transformar um vetor de *bytes* em uma *string* e vice-versa). Ler, alterar ou gravar uma nova linha em uma tabela do banco de dados é responsabilidade de outras classes.

5.4.1.1 Modelo do Usuário

O modelo de dados do usuário segue a especificação definida pelo OpenID [19]. Define-se nessa especificação uma quantidade mínima de informações sobre usuário para poder diferenciá-lo e o formato de como essas informações devem ser guardadas.

As informações sobre o usuário foram modeladas como uma tabela no banco de dados, onde cada usuário é representado por uma linha na tabela "Users", com os seguintes campos:

- *nickname* (apelido): *string* definida pela especificação [19]. A URI do usuário tem sempre o formato: "http://openid.gta.ufrj.br/NICKNAME". Ou seja, o apelido assim como o número de cadastro, deve ser único e é responsabilidade da classe "user" de detectar se um apelido é repetido ou não;

- e-mail: outra variável definida pela especificação, essa variável deve ter o formato definido pela RFC2822 [39]. A classe *InternetAddress* já implementa a funcionalidade de verificação em Java e foi utilizada;

- *fullname* (nome completo): definida na especificação;

- dob(*date of birth* – data de nascimento): definida na especificação com o formato AAAA-MM-DD;

- *gender*(gênero): definida na especificação;

- *postcode*(código postal): definido na especificação;

- *country*(país): definido na especificação, deve estar entre os países definidos pela ISO3166 [40];

- *language*(língua): definido na especificação, deve estar de acordo com o padrão ISO639 [41];

- *timezone*(fuso horário): definida na especificação, deve estar de acordo com o banco de dados TimeZone [42];

- *creationtime*: data e hora de criação do cadastro.

Outras três tabelas estão diretamente relacionadas com os usuários. Elas contêm as informações dos três padrões definidos acima (ISO3166, ISO639 e dados do *TimeZone*). Essas três tabelas devem ser criadas e preenchidas antes do *servlet* estar no ar. Um programa foi escrito para completar essas tabelas com os dados necessários, ele carrega as informações de arquivos-texto e coloca-as no banco de dados. Esse programa chama-se "dbPopulator".

5.4.1.2 Modelo de Cartões Inteligentes

Os dados sobre os cartões são diretamente ligados aos usuários. Os cartões inteligentes são divididos em tipos e cada cartão deve ter um dono que já tenha sido registrado no banco de dados. Essa portabilidade é garantida somente se houver bibliotecas em Java capazes de criptografar e decriptografar as mensagens trocadas com esses cartões. Essa discussão vai ser retomada na conclusão desse texto. Para essa implementação, como citado anteriormente, foi principalmente utilizado o cartão de modelo JCOP21 da NXP, que contém sistemas operacionais JavaCard 2.1.1.

Duas tabelas foram criadas no banco de dados para descrever esses cartões. A primeira é a tabela de tipos de totens, a "CardType". Essa tabela contém informações genéricas sobre um grupo de cartões, tais como nome do modelo de cartão ou do fabricante. A principal característica que separa os cartões é como o servidor pode fazer a interface com os cartões daquele grupo. A interface entre o cartão e o servidor tem diversas etapas. No registro, o servidor vai indicar qual a aplicação mais adequada a se instalar nesse cartão. Essa aplicação deve ser a mesma dentro de um mesmo grupo. Além disso, dois *java applets* foram desenvolvidos para esse cartão, um de registro e outro de autenticação. Ambos devem ser os mesmos para todos os cartões de um grupo. E por último, o servidor deve ser capaz de criptografar/decriptografar mensagens com esse tipo, e para isso, vai utilizar uma mesma classe para todos os totens de mesmo grupo. A tabela de tipos é composta pelos seguintes campos:

- *typeName*(nome do tipo): uma *string* que não pode ser repetida que informa qual o nome desse grupo de cartões;
- *manufacturer*(fabricante): nome do fabricante desse grupo;

- *pathToRegApplet*, *pathToAuthApplet*: caminhos para os *Java Applets* responsáveis por registrar e autenticar o usuário com um cartão desse grupo;
- *className*: nome da classe que herda da interface "SCReader" e que seja capaz de (de)criptografar mensagens;
- pubKey: quatro *strings* que podem ser usadas para guardar a chave pública do servidor, para esse grupo. No caso dessa implementação, o algoritmo utilizado foi o RSA, onde uma *string* guarda o módulo e outra guarda o expoente da chave pública;
- *privKey*: quatro *strings*, tal como a *pubKey*, para guardar a chave privada do servidor;
- *creationtime*: data e hora de criação do cadastro.

A segunda tabela é a "UserCards" e guarda dados específicos de cada cartão, no caso dessa implementação, a tabela possui campos específicos para cartões inteligentes. A tabela é composta por:

- *userID*: numero identificador do usuário que possui esse cartão;
- *typeName*: nome do grupo ao qual esse cartão pertence;
- AID(*Application Identifier* – Identificador de Aplicação): código do aplicativo instalado dentro do cartão. Quando a comunicação entre o cartão e o computador é aberta, o *java applet* tem ainda que escolher qual aplicativo dentro do cartão vai ser executado. Lembrando que em certos cartões JavaCard, mais de um aplicativo pode estar instalado no cartão;
- *pubKey*: *strings* que representam a chave pública do cartão;
- ATR(*Answer To Reset*): quando um cartão é inicializado, este envia um código de resposta. Esse código carrega informações sobre o fabricante o modelo do cartão. Esse vetor de bytes é estocado no banco de dados durante o registro do cartão;
- *creationtime*: data e hora de criação do cadastro.

5.4.1.3 Modelo de Sessões

Informações sobre as sessões podem ser guardadas dentro do banco de dados. Esses dados permitem identificar o usuário, o cartão usado e quais provedores de serviços em que o usuário se autenticou pelo servidor OpenID. Essas informações permitem manter um dossiê sobre o comportamento do usuário e podem ser informações valiosas em caso de roubos de identidades.

O banco de dados possui duas tabelas para guardar as informações de sessões. Ambas têm os mesmos dados. Uma tabela ("Sessions") guarda dados sobre as sessões atualmente abertas enquanto "finishedSessions" guarda dados sobre as sessões passadas. As informações são:

- Id: identificador de sessão. Uma *string* criada pelo próprio Tomcat e que nunca se repete entre as sessões abertas;
- *Statuscode*: status da sessão. Esse status vem de SessionStatus, uma classe feita em Java para esse projeto que contém todos os estados possíveis das respostas do servidor OpenID;
- *userKey*: número de cadastro do usuário que abriu essa sessão;
- *cardKey*: cartão usado pelo usuário durante essa sessão;
- *assocHandle*: código da associação entre o provedor de serviços (SP) e o servidor OpenID para essa sessão;
- *creationtime*: hora e data da criação;
- *lastchange*: hora e data da última alteração na sessão;
- *finishtime*: esse dado só existe na tabela "finishedSession" e indica a hora e data em que a sessão foi terminada.

Cada usuário deve se autenticar toda vez que é redirecionado para o provedor de serviços. Isso evita que um atacante entre em um serviço como um determinado usuário, logo após desse usuário ter se autenticado. Os dados sobre as sessões são guardados para manter o registro do que foi feito por um determinado usuário.

5.4.1.4 Modelo de Associação

Os provedores de identidade e de serviços podem estabelecer um segredo compartilhado que vai ser utilizado nas assinaturas das mensagens HTTP. Esse segredo é definido em uma associação entre os provedores e pode ser utilizado em mais de uma operação de autenticação entre esses provedores. Da mesma forma como dados sobre as sessões de usuários são mantidos, informações sobre as associações entre o servidor OpenID e provedores de serviços (SPs) também são guardados.

As associações são modeladas de forma parecida com as sessões, onde duas tabelas foram criadas: "associations" para associações que ainda estão válidas, e quando elas expiram, são transferidas para "expiredAssociations". As associações são descritas assim:

- rpURL: endereço web do provedor de serviços que iniciou a associação;
- *expireIn*: duração em que a associação é válida, representada como um inteiro;
- *keyType*: qual é o tipo de chave (Diffie-Hellman, por exemplo) que foi usada para fazer a associação;
- *mackey*: segredo compartilhado entre os provedores. Esse segredo é utilizado em um *hash* com os campos que devem ser assinados na mensagem HTTP;
- *creationtime*: hora e data da criação;
- *expirationtime*: esse campo existe somente na tabela "expiredAssociations" e guarda a hora e data da expiração da associação.

5.4.2 Interface

A comunicação com o usuário é feita pela Internet, onde o usuário tem acesso às interfaces do servidor pelo seu navegador. O servidor OpenID possui páginas que permitem ao usuário ler e alterar suas informações de cadastro, tal como foi discutido anteriormente. A página principal é mostrada na Figura 5.5.

O usuário pode escolher entre se autenticar e acessar seus dados de cadastro ou se registrar. Usuários que desejem se autenticar devem fornecer o seu apelido (*nickname*) e são reenviados para uma página contendo HTML o *Java Applet* capaz

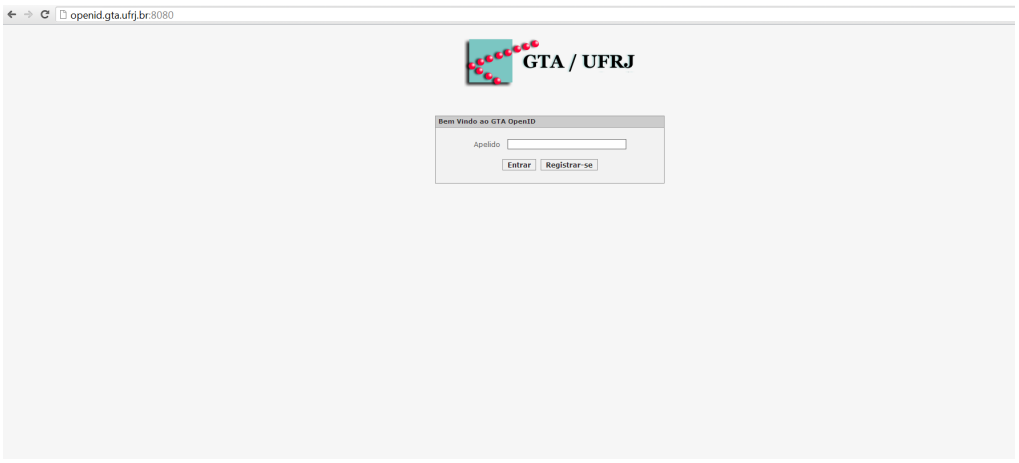

Figura 5.5: Página de Login.

de fazer a comunicação entre o cartão e o servidor. Após a autenticação, os usuários têm acesso aos seus dados de cadastro.

Figura 5.6: Página de Dados.

Os usuários podem modificar seus dados e ver o URI atribuído pelo servidor a ele (no caso, http://openid.gta.ufrj.br:8080/server/teste). Esse URI pode ser utilizado em provedores de serviços e serão redirecionados para a página de autenticação com o *Java Applet*. Assim que a autenticação terminar, o usuário é redirecionado de volta para o provedor de serviços.

A parte de registro é dividida em duas seções. Quando o usuário decide se registrar, ele deve primeiramente criar uma conta com seus dados pessoais e em seguida cadastrar seu cartão no servidor. A interface de escolha é mostrada na Figura 5.7:

Onde o usuário pode escolher entre se cadastrar no servidor ou registrar o

Figura 5.7: Página para escolher registro.

seu cartão. Na escolha de registrar o cartão, o mesmo é redirecionado para o site com um *Java Applet*, que estabelece a comunicação entre o cartão e o servidor e os dados são trocados.

5.4.3 Controle

O Controle pode ser definido como as requisições HTTP enviadas pelo navegador do usuário. Essas mensagens podem ser tanto o usuário que efetuou alguma entrada no navegador quanto os *Java Applets* trocando mensagens com o servidor para registrar ou autenticar um cartão.

5.4.4 Detalhes de Segurança da Implementação

A implementação específica para testar esse projeto é baseada em três peças fundamentais: o servidor web Tomcat 7, o banco de dados em MySQL Server, o sistema operacional Debian 6. O objetivo dessa seção é levantar os principais cuidados com a segurança dessa implementação.

Duas questões são centrais na segurança do projeto:

- acesso de *Java Applets* ao leitor de cartões: *Java Applets* são executados através de navegadores em máquinas virtuais especiais conhecidas como "sandboxes". Essas máquinas virtuais não têm permissão de acessar o *hardware* ou executar programas por questões de segurança. Para permitir esse tipo de acesso, os *Applets* devem ser digitalmente assinados e os usuários são avisados, geralmente através de uma *pop-up* sobre os riscos. Uma melhoria de se-

gurança nesse caso é sempre usar um canal SSL quando enviar *Java Applets*, o que confirma a identidade do servidor OpenID através de um certificado digital;

- registro de cartões: como discutido anteriormente, a opção de registrar remotamente cartões foi deixada em aberto nessa implementação. Qualquer usuário já cadastrado no sistema pode registrar o seu cartão. Os riscos são: primeiro de atacantes iludirem usuários a registrarem seus cartões em servidores falsos e ficarem suscetíveis a ataques de *phishing* no registro de cartões. Outra possibilidade é um ataque de negação de serviços onde o atacante registra diversas contas e chaves de cartões falsos no servidor. Cada ataque levando a uma operação de acréscimo de uma nova linha na tabela do banco de dados. A melhor alternativa contra o *phishing* no registro de cartões é o uso de canais SSL para comprovar a autenticidade do servidor. Já existem propostas para proteção contra ataques de negação de serviços e elas vão além do escopo desse projeto.

Capítulo 6

Resultados e Conclusão

O gerenciador de identidades implementado é eficiente e seguro. A eficiência do gerenciador de identidades é porque o OpenID é um protocolo leve que não exige muito esforço do usuário. Além disso, as mensagens são definidas no formato HTTP e, portanto, o usuário precisa apenas de um navegador. O outro ponto central desse projeto foi segurança na autenticação. Para garantir uma autenticação confiável, utilizam-se cartões inteligentes com alto nível de segurança. Deve ser ressaltado que o ganho obtido em segurança pelo uso dos cartões inteligentes não acarreta maiores dificuldades para o usuário, pois todo o procedimento é feito por *Java Applets* em seu navegador.

O projeto consistiu da implementação de um sistema de gerenciamento de identidades de acordo com as especificações [16] [19]. Além dessa implementação, neste capítulo é feita uma análise de segurança do sistema levando em consideração as falhas de segurança já conhecidas no OpenID, como citado em [16] [29] [43]. Para essa análise, os modelos de atacante considerados são descritos a seguir, assim como técnicas para se prevenir esse ataque.

Apesar do gerenciamento de identidades estar separado em três entidades diferentes, as ligações entre essas entidades podem compartilhar muitos caminhos em comum. Considerando esse cenário, um atacante pode mais facilmente interceptar mensagens trocadas entre todas as entidades ou simular duas entidades para ataques mais sofisticados.

A especificação do OpenID definido o protocolo de troca de mensagens HTTP. Isto não evita que todas as mensagens sejam "bisbilhotadas" (*eavesdropping*). Uma forma de evitar que as mensagens sejam inspecionadas é o uso do protocolo HTTPS

que se serve de canais seguros como SSL ou TLS. No entanto, o ataque *SSLStrip* redireciona o usuário da porta HTTPS para a HTTP, evitando assim a construção do túnel seguro.

6.1 Ataques ao Usuário

Nessa seção, discutem-se ataques ao agente de usuário ou ao sistema operacional no qual esse agente é executado. Os ataques mais convencionais que objetivam roubar identidades a partir do computador do usuário são o *keylogging* e o *phishing*. No primeiro, o atacante consegue instalar um programa no computador alvo que seja capaz de ler todas as suas entradas de teclado e/ou mouse. O *phishing* é um ataque que procura "pescar" informações relevantes do usuário, como por exemplo as credenciais de autenticação. No *phishing* um sítio fraudado eletronicamente procura enganar o usuário fazendo-se passar pelo sítio confiável. No caso do protocolo OpenID, o atacante simula um provedor de serviços (SP) falso e esse envia o usuário para provedor de identidades (OP) falso. Nesse sítio o usuário entra suas credenciais. Esse tipo de ataque é muito comum em roubo de dados bancários. Ambos os ataques têm como objetivo de roubar as credenciais de autenticação que o usuário conhece, ou seja, a senha. A melhor defesa é eliminar a dependência de senhas. A proposta deste projeto é utilizar microcontroladores seguros capazes de executar algoritmos de criptografia assimétrica [20]. Nesse tipo de arquitetura, a autenticação é do tipo o que o usuário possui e não apenas do que ele conheça. Assim, ficam impossibilitados ataques a senhas como o *keylogging* e o *phishing*.

Outro ataque é por um vírus de computador capaz de inspecionar a memória do computador. Caso a senha ou a chave criptográfica usada na autenticação passe para a memória, o vírus pode roubá-la. Esse tipo de ataque não é possível nesse projeto pois tanto as chaves quanto o processamento dos desafios do protocolo de autenticação são feitos dentro do cartão e do servidor OpenID, ambos considerados confiáveis.

6.2 Ataque de Escuta e Reutilização

Um atacante pode escutar todas as mensagens trocadas entre as entidades, capturando a mensagem de asserção positiva, onde o usuário é redirecionado do

provedor de identidades de volta para o provedor de serviços e é aceito no serviço. O ataque consiste em capturar essa última mensagem e reutilizá-la para ser autenticado. Para evitar esse ataque as mensagens devem conter números utilizados uma única vez, como por exemplo, *timestamps* ou *nonces*. Além disso, esses elementos devem ser sempre incluídos na assinatura da mensagem HTTP para garantir a sua integridade. Uma vez que a especificação 2.0 do OpenID já coloca este procedimento como exigência, o projeto implementou *nonces* assinados nas mensagens de autenticação.

6.3 Ataques de Interposição - *Man-In-The-Middle*

Existem diversos tipos de ataques de interposição. Onde o atacante forja a sua atuação como uma ou mais entidades do protocolo. Os modelos de ataques mudam conforme o objetivo final. O primeiro a ser considerado tem como objetivo autenticar um atacante como um usuário válido. Uma possibilidade é vista em [43], onde o atacante forja ser um usuário e um provedor de identidades ao mesmo tempo.

1. O atacante contata o provedor de serviços com o contato de um determinado usuário;

2. O provedor de serviços (SP) tenta fazer a descoberta do provedor de identidades e o atacante responde;

3. O SP faz o pedido de autenticação e o atacante ganha acesso ao serviço;

4. Atacante entra no lugar do usuário dentro do provedor de serviços.

Esse ataque é possível na especificação OpenID, pois todas as mensagens de autenticação passam pelo usuário. O que lhe permite agir tanto como usuário quanto provedor de identidades, já que a associação é possível, mas não obrigatória. Uma vez que a associação é uma comunicação direta entre os provedores, e iniciada pelo SP, o atacante precisa executar uma ação mais sofisticada, como alterar a resolução do DNS (DNS *Spoofing*). Nesses ataques, o SP encaminharia os pacotes para o OP falso. A solução desse gênero de ataque é obrigar o uso de associações entre os provedores através de canais seguros, como por exemplo, canais SSL/TLS).

Esses canais obrigam que as duas partes apresentem certificados assinados válidos e, portanto, tornam um ataque muito mais complexo.

6.4 Ataques de Negação de Serviço

Existem diversos tipos de ataques de negação de serviços no OpenID. Eles podem ter como alvo provedores de serviços ou de identidades. Para esse projeto foi considerado os ataques a provedores de identidades. Ataques de negação de serviço do OpenID são citados em [16]. Os ataques possíveis são:

- por associações: o ataque é mais sofisticado, onde o provedor de identidades primeiramente recebe um pedido de autenticação, e um pedido de associação. O pedido de associação é dividido em duas mensagens onde o provedor de serviços envia a sua chave Diffie-Hellman para o provedor de identidades OpenID (OP) e esse calcula a chave secreta para a assinatura. O ataque de negação acontece, pois cada pedido de associação obriga o provedor de identidades a efetuar operações para calcular o algoritmo de Diffie-Hellman;

- por registro: um ataque de negação onde diversos pedidos de registro são enviados para o provedor de identidades, obrigando-o a fazer operações de verificação sobre os dados recebidos e no banco de dados para verificar se já existe ou não uma conta repetida e gravá-la;

- por registro de cartão: esse ataque de negação é particular a essa implementação, onde o usuário pode registrar cartões remotamente. O atacante pode simular cartões inteligentes e registrar chaves RSA no servidor, o que causa os mesmos problemas do caso anterior. As soluções para esse caso já foram discutidas no capítulo anterior;

- por pedidos de autenticação: em provedores de identidades que utilizam credenciais de autenticação como senhas, esse é o ataque de negação de serviços mais simples e fácil de tratar. No entanto, nesse projeto um atacante pode simular o funcionamento de um cartão inteligente e efetuar operações de autenticação falsas. O ataque obriga o servidor OpenID a efetuar operações no banco de dados e de criptografia assimétrica;

As soluções de ataques de negação de serviços são complexas e vão além do escopo desse projeto.

6.5 Manipulação de Parâmetros

Em [29] um novo padrão de ataques foi abordado. Esse padrão descreve ataques às mensagens definidas no OpenID sem envolver nenhuma das entidades. Esse padrão pode ser dividido em dois tipos diferentes de ataques, ambos visando manipular os parâmetros das mensagens:

- Adulteração de Parâmetros: um atacante altera parâmetros em uma mensagem;
- Injeção de Parâmetros: um atacante insere parâmetros não definidos em uma mensagem;

Ambos os tipos de ataques têm pouca capacidade de danos, podendo principalmente alterar a experiência do usuário durante a utilização do serviço. O principal risco é alterar de alguma forma o e-mail de um usuário. Uma vez que e-mails são usados por serviços para alertar usuários sobre o que acontece com suas contas. Alterar o e-mail de alguém permitiria a um atacante poder acompanhar as ações de usuários dentro do serviço.

Existem diversas formas de confrontar esse tipo de ataque. O OpenID prevê que as mensagens carreguem assinaturas para identificar o provedor de identidades. Essas assinaturas englobam todos os parâmetros centrais, garantindo também a integridade destes. Uma solução é estender essa assinatura para todos os parâmetros. Outra possibilidade é sempre utilizar canais de comunicação seguros, tais como SSL ou TLS, entre o provedor de serviços (SP) e o usuário e o provedor de identidades OpenID (OP) e esse usuário.

6.6 Ataques de Sessão

Alguns ataques se aproveitam da sessão aberta entre o usuário e o provedor de identidades. O mais conhecido pode ser visto em [31], o *Cross-Site-Request-Forgery*. Alguns provedores de identidades pedem a autenticação do usuário uma única vez.

Portanto, assim que uma sessão foi aberta no provedor de identidades OpenID (OP) para um determinado serviço, um atacante pode se declarar como sendo esse usuário e ganhar acesso a outro serviço sem precisar se autenticar.

A solução adotada nesse projeto é obrigar a autenticação do usuário sempre que este for acessar um serviço e for redirecionado. Isso evita que um atacante se aproveite de uma sessão já aberta e não causa muitos transtornos para usuário além de ser redirecionado para o provedor de identidades, uma vez que o usuário precisa apenas inserir o cartão inteligente novamente.

Capítulo 7

Conclusão e Trabalhos Futuros

Identificação é um tema importante na Internet. Atualmente, o controle de identidades digitais é feito serviço a serviço ou provedor a provedor. No entanto, diversos estudos e especificações tentam resolver esse problema. O OpenID é um protocolo que foi definido para solucionar alguns desses problemas.

O maior problema do gerenciamento de identidades é a sua segurança. Uma vez que roubar uma identidade agora significa ter acesso a todos os serviços que um determinado usuário utiliza. Um sistema de gerenciamento de identidades deve garantir um processo seguro de autenticação e de comunicação entre o provedor de serviços e o usuário. Por conta desse problema, microcontroladores seguros com a capacidade de executar algoritmos de criptografia são utilizados.

O objetivo deste projeto é criar um gerenciador de identidades que utilize cartões inteligentes (*smart cards*). Nesse projeto, o recurso que exerce a função do conhecido apelido e senha da Internet é o par de chaves do algoritmo RSA. Onde a chave privada é guardada de forma segura dentro da memória do microcontrolador no cartão, que é considerada inviolável. A grande vantagem, além de proteger a chave, é a facilidade para usuário, que não precisa mais memorizar uma senha complicada para se autenticar. A sua segurança foi testada de acordo com as especificações do OpenID além de outros estudos publicados sobre o tema. Novas falhas de segurança decorrentes de detalhes da implementação também foram levados em consideração. A implementação da proposta se mostrou robusta e eficiente. Atualmente, esse servidor OpenID faz parte do Testbed para Internet do Futuro, o FITS – *Future Internet Testebed with Security* , uma plataforma de testes inter-universitária para se experimentar e novos protocolos de redes para a Internet. O servidor OpenID

desenvolvido é utilizado como provedor de identidades nesse ambiente.

Josang e Pope discutem em [5] um sistema de gerenciamento de identidades onde as credenciais podem ser guardadas dentro de PADs(*Personal Authentication Devices*). Urien, Marie e Kiennert propõem [32] um sistema de gerenciamento utilizando o OpenID, onde o cartão inteligente pode guardar identidades do usuário. A proposta de trabalho futuro é adaptar esse projeto para um sistema centrado no usuário. Os dados de um determinado usuário podem ser guardados dentro de um microcontrolador seguro. A responsabilidade do provedor de identidades nesse caso é de autenticar corretamente o usuário. Outra direção promissora é a incorporação de métodos para proteger a privacidade do usuário. O OpenID não é especializado em proteger os dados de seus usuários e, portanto, uma proposta de trabalho futuro é de estudar e implementar técnicas para garantir essa proteção nesse projeto.

Como trabalho futuro pretende-se criar uma plataforma mais versátil de gerenciamento de identidades, em que um usuário possa utilizar mais tipos diferentes de microcontroladores seguros. A proposta é incorporar TPMs como no sistema de gerenciamento de identidades atual e avaliar as vantagens e desvantagens do *java cards* e do TPMs. Outra área estratégica do grupo de pesquisa do GTA é o estudo de autorizações para a administração de redes virtuais dentro do FITS.

Referências Bibliográficas

[1] BERTINO, E., TAKAHASHI, K., *Identity Management: Concepts, Technologies and Systems*. 685 Canton Street Norwood, MA, United States, Artech House, 2010.

[2] *NGN identity management framework*. International Telecommunication Union, 2009.

[3] CAMERON, K., "The Laws Of Identity", http://www.identityblog.com/stories/2004/12/09/thelaws.html, 2005.

[4] WANGHAM, M. S., MELLO, E. R., BORGER, D. S., *et al.*, *Minicursos do X Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, chapter Gerenciamento de Identidades Federadas, Porto Alegre, SBC, pp. –, 2010.

[5] JØSANG, A., POPE, S., "User centric identity management". In: *AusCERT Asia Pacific Information Technology Security Conference*, p. 77, Citeseer, 2005.

[6] DHAMIJA, R., DUSSEAULT, L., "The Seven Flaws of Identity Management - Usability and Security Challenges", *IEEE Security & Privacy*, pp. 24–29, 2008.

[7] MALER, E., REED, D., "The Venn of Identity - Options and Issues in Federated Identity Management", *IEEE Security & Privacy*, pp. 16–23, 2008.

[8] PFITZMANN, A., HANSEN, M., "Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonimity and Identity Management - A Consolidated Proposal for Terminology", http://dud.inf.tu-dresden.de/Anon_Terminology.shtml.

[9] LEVY, S., GUTWIN, C., "Improving understanding of website privacy policies with fine-grained policy anchors". In: *Proceedings of the 14th international conference on World Wide Web*, pp. 480–488, ACM, 2005.

[10] RAGOUZIS, N., HUGHES, J., PHILPOTT, R., *et al.*, "Security Assertion Markup Language (SAML) V2.0 Technical Overview", https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf.

[11] LAWRENCE, K., KALER, C., NADALIN, A., *et al.*, "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[12] GUDGIN, M., HADLEY, M., MENDELSOHN, N., *et al.*, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)", http://www.w3.org/TR/soap12-part1/, 2007.

[13] CHAPPELL, D., "Introducing Windows CardSpace", http://msdn.microsoft.com/en-us/library/aa480189.aspx#introinfocard_topic2, 2006.

[14] DAVIS, D., MALHOTRA, A., WARR, K., *et al.*, "Web-Services Metadata Exchange (WS-MetadataExchange)", http://www.w3.org/TR/ws-metadata-exchange/, 2011.

[15] JRSTAD, I., JOHANSEN, T., BAKKEN, E., *et al.*, "Releasing the potential of OpenID & SIM". In: *Intelligence in Next Generation Networks, 2009. ICIN 2009. 13th International Conference on*, pp. 1 –6, 2009.

[16] FITZPATRICK, B., RECORDON, D., "OpenID Protocol 2.0", http://openid.net/specs/openid-authentication-2_0.html, 2005.

[17] REHMAN, R. U., *Get Ready for OpenID*. Conformix Books, 2007.

[18] HARDT, D., BUFU, J., HOYT, J., "OpenID Attribute Exchange Extension 1.0", http://openid.net/specs/openid-attribute-exchange-1_0.html.

[19] HOYT, J., DAUGHERTY, J., RECORDON, D., "OpenID Simple Registration Extension 1.0", http://openid.net/specs/openid-simple-registration-extension-1_0.html.

[20] BUFU, J., DAUGHERTY, J., "OpenID Provider Authentication Policy Extension 1.0", http://openid.net/specs/openid-provider-authentication-policy-extension-1_0.html.

[21] WACHOB, G., REED, D., CHASEN, L., *et al.*, "Extensible Resource Identifier (XRI) Resolution Version 2.0", http://docs.oasis-open.org/xri/2.0/specs/cd02/xri-resolution-V2.0-cd-02.html, 2007.

[22] MILLER, J., "Yadis Specification 1.0", http://openid.net/specs/openid-authentication-2_0.html#rfc.references1.

[23] BUCHMANN, J. A., *Introduction to Cryptography*. Springer, 2004.

[24] KAUFMAN, C., PERLMAN, R., SPECINER, M., *Network security: private communication in a public world*. Prentice Hall, 2002.

[25] RANKL, W., EFFING, W., *Smart Card Handbook*. Wiley.

[26] CHEN, Z., *Java Card Technology for Smart Cards - Architecture and Programer's Guide*. One Lake Street, Reading Massachusetts 01867, Pearson, 2000.

[27] "FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION - SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES", http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

[28] "FIPS140-2 Security Policy for the JCOP21id 32K JavaCard Platform", http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp363.pdf.

[29] VAN DELFT, B., OOSTDIJK, M., "A Security Analysis of OpenID". In: de Leeuw, E., Fischer-Hübner, S., Fritsch, L. (eds.), *Policies and Research in Identity Management*, v. 343, *IFIP Advances in Information and Communication Technology*, Springer Boston, pp. 73–84, 2010.

[30] "Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM)", http://www.ietf.org/rfc/rfc4186.txt.

[31] LEICHER, A., SCHMIDT, A. U., Y., S., "Smart OpenID A Smart Card Based OpenID Protocol". In: *IFIP Advances in Information and Communication Technology*, v. 376, pp. 75–86, 2012.

[32] URIEN, P., MARIE, E., KIENNERT, C., "A New Convergent Identity System Based on EAP-TLS Smart Cards". In: *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pp. 1–6, may 2011.

[33] SIMON, D., ABOBA, B., HURST, R., "The EAP-TLS Authentication Protocol", http://www.ietf.org/rfc/rfc5216.txt.

[34] URIEN, P., "Collaboration of SSL smart cards within the WEB2 landscape". In: *Collaborative Technologies and Systems, 2009. CTS '09. International Symposium on*, pp. 187–194, may 2009.

[35] LEICHER, A., SCHMIDT, A., SHAH, Y., *et al.*, "Trusted Computing Enhanced OpenID". In: *Internet Technology and Secured Transactions (ICITST), 2010 International Conference for*, pp. 1–8, nov. 2010.

[36] "TCG Infrastructure Working Group Reference Architecture for Interoperability(Part 1)", http://www.trustedcomputinggroup.org/files/resource_files/8770A217-1D09-3519-AD17543BF6163205/IWG_Architecture_v1_0_r1.pdf.

[37] AKRAM, R., MARKANTONAKIS, K., MAYES, K., "User Centric Security Model for Tamper-Resistant Devices". In: *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on*, pp. 168 –177, oct. 2011.

[38] "OpenID4Java", http://code.google.com/p/openid4java/.

[39] RESNICK, P., "Internet Message Format", http://www.ietf.org/rfc/rfc2822.txt.

[40] "ISO3166 - Lista de Países", http://www.iso.org/iso/country_names_and_code_elements.

[41] "ISO 639 Language Codes", http://www.w3.org/WAI/ER/IG/ert/iso639.htm.

[42] "Sources for Time Zone and Daylight Saving Time Data", http://www.twinsun.com/tz/tz-link.htm.

[43] LINDHOLM, A., *Security Evaluation of OpenID Protocol*. M.Sc. dissertation, Royal Institute of Technology - KTH, 2009.

Apêndice A

Diagrama de Classes

A Figura A.1 mostra o diagrama das principais classes da implementação, bem como as suas interações. A classe principal na Figura A.1 é a "OPServer". Essa classe implementa o *servlet*, que recebe e trata todos os pedidos HTTP. Sua tarefa é receber uma mensagem HTTP, criar o objeto que trata apropriadamente a essa requisição e enviar a resposta em formato HTTP. Um exemplo: quando um usuário faz o seu cadastro no site, o *servlet* cria um objeto "user", responsável por verificar se as entradas do usuário estão de acordo com os padrões e, caso afirmativo, gravar no banco de dados. As outras classes são:

- *user* - essa classe representa um usuário de acordo com os seus parâmetros, guardados no banco de dados. Ela é responsável por editar, verificar e gravar esses parâmetros;
- *register* - classe responsável por registrar um novo usuário no banco de dados, verificando se seus parâmetros estão de acordo com a especificação;
- *SCReader* - interface que define os métodos de comunicação entre um cartão e o servidor;
- *SCReaderFactory* - essa classe recebe um pedido de autenticação ou registro de cartão e aloca o *SCReader* correspondente. Ao ser requisitada, essa classe retorna um HTML que permite ao usuário escolher o tipo de cartão que deseja autenticar. No caso dessa implementação, apenas um leitor de cartões foi implementado, o *JCOP2136Reader*;
- *JCOP2136Reader* - essa classe implementa os métodos definidos em *SCReader* e é responsável por fazer a comunicação entre o servidor e cartões JCOP21 utilizados no projeto.

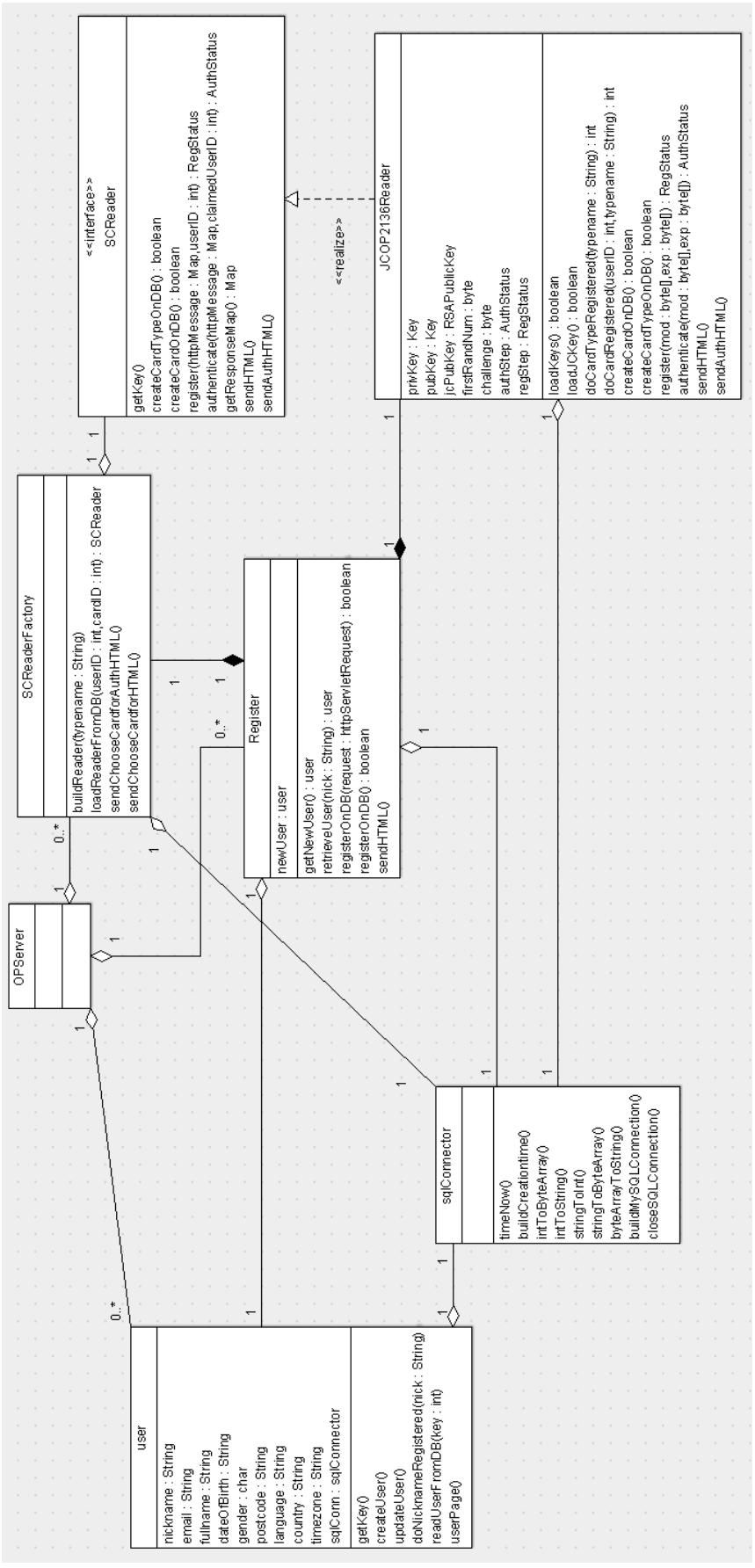

Figura A.1: Diagrama das Principais Classes.

Apêndice B

Ambiente de Programação do Cartão

Esse apêndice discute o ambiente de programação e de testes do *java card* para esse projeto, além de documentar o código da aplicação implementada no *java card JCOP21*. Esse código executa toda a comunicação do cartão com o servidor. Algumas das funções implementadas têm como objetivo testar os valores das chaves criadas pela aplicação e podem ser retiradas sem prejuízo do funcionamento do cartão.

B.1 Ambiente de Programação e Testes

A aplicação para o cartão foi desenvolvida em duas partes. Primeiro, uma aplicação foi criada e testada no simulador da API do cartão. O teste desenvolvido para essa aplicação foi executar operações de criptografia com o simulador. Um programa capaz de comunicar com o simulador e receber entradas do cartão foi desenvolvido. Esse programa contém um módulo capaz de decriptografar as mensagens do cartão e criptografar com o seu próprio par de chaves RSA.

O simulador do cartão é chamado de *Java Card Workstation Development Environment (JCWDE)* e pode simular o seu funcionamento. Quando o simulador é inicializado, ele carrega uma aplicação compilada para *java card* e abre uma conexão TCP de acesso local. Um programa pode se comunicar com o simulador conectando-se e enviando mensagens APDU. Uma vantagem desse simulador é a sua integração com IDEs, como Eclipse.

O cartão escolhido foi o JCOP21 com um leitor e gravador SCR 3310 CIS com comunicação USB para computadores. Uma vez que a aplicação foi simulada e testada no JCWDE, o próximo passo foi gravá-la no cartão. Para isso, utilizou-se uma ferramenta open-source chamada "GPShell". Essa ferramenta é capaz de estabelecer uma comunicação segura com o cartão, carregar, instanciar, deletar e listar aplicações em cartões suportados. A aplicação exigiu algumas alterações para poder ser instalada. Em seguida, foi testada com o programa desenvolvido anteriormente.

B.1.1 Memória do Cartão

Existem três tipos de memórias no cartão inteligente: ROM, RAM e EEPROM. A memória ROM pode ser escrita uma única vez e isso é feito durante a produção do cartão. A memória EEPROM é muito mais lenta que a RAM, mas

quando o cartão é desligado, o conteúdo continua preservado. As aplicações são objetos do tipo *Applet* e são gravados dentro da EEPROM. Durante o processo de instalação de uma aplicação alocam-se os objetos na EEPROM. Todos os objetos alocados com o operador "new" são gravados na EEPROM. Recomenda-se que esses objetos sejam alocados durante a instalação da aplicação, pois nem todos os cartões permitem alocação dinâmica na EEPROM após a instalação. Nem todos os cartões oferecem a funcionalidade de "garbage colector".

B.1.1.1 Objetos Persistentes e Transientes

Existem dois tipos de objetos na API Java Card:

- objetos persistentes - são alocados dentro da EEPROM e não são apagados caso o cartão seja desconectado;
- objetos transientes - são alocados na memória RAM e são reiniciados para quando o cartão é desconectado.

Objetos persistentes são alocados com o operador "new". Nem todos os cartões permitem que objetos sejam alocados fora da instalação da aplicação. Objetos transientes porém podem ser alocados em qualquer momento do código. Geralmente, um objeto persistente aponta para objetos transientes.

Objetos transientes são alocados em qualquer momento do código, através de métodos especiais, definidos no pacote *JCSystem*. Um exemplo de método é o *public static byte[] makeTransientByteArray(short length, byte event)*. Esses objetos são reinicializados quando um determinado evento definido pelo *byte event* acontece (exemplo: o objeto é reiniciado quando a aplicação é deselecionada).

Algumas operações especiais são definidas para se alterar os dados de objetos transientes. O objetivo é garantir que as mudanças em objetos sejam todas feitas ao mesmo tempo. Caso haja falha no cartão ou no código, os valores não vão ser atualizados. Essas funções são utilizadas como em bancos de dados, onde primeiro abre-se a transação com *beginTransaction()*, as alterações são declaras e finalmente se fecha com *commitTransaction()*.

B.2 Código

Todas as aplicações para *java card* devem ter seu *package*. Para esse projeto, foi criado o *package JCCipher*. Ele importa os principais elementos da API Java Card: *javacard.framework.**, que é responsável pelos principais métodos, como o *Applet* ou as classes que contém chaves e pela comunicação entre a *Java Card Run Environment(JCRE)* e a aplicação. Além disso, utiliza-se o *javacardx.crypto.Cipher*, que contém os métodos de comunicação com os coprocessadores de criptografia do cartão. Enquanto que o *javacard.security.** implementa as principais classes de chaves criptográficas e métodos de geração de números aleatórios.

Um dos principais métodos do JCCipher é o *loadReceivedDataToBuffer(APDU arg0)*, responsável por receber todo o APDU de comando e gravar seus dados dentro de um *buffer*. Esse método vai ser utilizado durante toda a aplicação.

```
1 //Load the arriving data from the APDU command to the buffer array.
2 public short loadReceivedDataToBuffer(APDU arg0) {
3   byte[] apdu = arg0.getBuffer();
```

```
4     short bytesRead = arg0.setIncomingAndReceive();
5     short echoOffset = (short)0;
6     while ( bytesRead > 0 ) {
7      Util.arrayCopyNonAtomic(
8        apdu,                  //Data Source
9        ISO7816.OFFSET_CDATA,//Start Point
10       buffer,                //Destiny Point
11       echoOffset,            //Start Point
12       bytesRead);            //Length of Bytes for copying
13     //shifts OffSet
14     echoOffset += bytesRead;
15     //Search for the Remaining Quantity
16     bytesRead = arg0.receiveBytes(ISO7816.OFFSET_CDATA);
17    }
18    return echoOffset;
19   }
20 }
```

O código da aplicação:

```
1  package jccipher;
2
3  import javacard.framework.APDU;
4  import javacard.framework.Applet;
5  import javacard.framework.ISOException;
6  import javacard.framework.JCSystem;
7  import javacard.framework.ISO7816;
8  import javacard.framework.Util;
9  import javacard.security.*;
10 import javacardx.crypto.Cipher;
```

A classe principal do projeto é a *JCCipher* estende a *Applet*. Apesar do nome, um *Java Card Applet* não tem relação com o applets de navegadores feitos em Java. Um *Applet* no cartão significa que o sistema operacional pode invocar essa aplicação e executá-la para receber, tratar e responder APDUs, sob ordens do usuário. Os *Applets* devem ter um número de registro no sistema operacional, chamado *AID - Application Identifier*. Quando um usuário deseja invocar o *JCCipher*, ele envia uma APDU específica contendo o AID para o JCRE, que aloca a aplicação correspondente. Esse comando é conhecido como *SELECT APDU*.

Os principais métodos do *Applet são*:

- *boolean select()* - retorna *true* se o comando recebido uma APDU de seleção. Geralmente, um comando de seleção é recebido pelas aplicações ativas antes da aplicação ser realmente selecionada;
- *void process(APDU apdu)* - toda vez que uma aplicação recebe uma APDU, o sistema operacional chama esse método. O método não retorna nenhum valor, mas é responsável por contruir a APDU de resposta;
- *void install()* - método invocado durante a instalação da aplicação no cartão. Esse método deve chamar o construtor da aplicação e registrá-la no sistema operacional;
- *void register()* - método utilizado para registrar o *Applet* no cartão com o identificador AID.

```
11
12
13 /**
14  * This applet is the responsable for handling the authentication
        process with an OpenID server
15  * that accepts the Java Server described for this project.
16  *
17  * @author Pedro Henrique Valverde Guimaraes
18  * @version 1.0
19  *
20  */
21
22 public class JCCipher extends Applet {
```

Os principais objetos são declarados aqui. Estes são vetores de bytes e chaves que vão ser utilizadas no restante do código. Uma atenção especial para os seguintes objetos: rsa_priv_key deve ser do tipo *RSAPrivateCrtKey*. Na API do Java Card, existem duas classes que podem representar a chave privada RSA, mas nem todas as duas são sempre implementadas. No caso do ambiente no JCOP21, apenas a classe *RSAPrivateCrtKey* é implementada. Nessa classe, a chave privada é representada de acordo com o CRT (*Chinese Remainder Theorem*). A chave pública do servidor OpenID é guardada em rsa_pub_host_key.

```
23  RSAPrivateCrtKey  rsa_priv_key;
24  RSAPublicKey rsa_pub_key;
25  RSAPublicKey rsa_pub_host_key;
26
27  Cipher cipher;
```

Durante o cálculo da APDU de resposta, valores intermediários precisam ser guardados. Esses valores são mantidos dentro do vetor de bytes *buffer*.

```
28  byte[] buffer;
```

Os vetores *userId* e *cardId* guardam os números de cadastro desse usuário e cartão, respectivamente. Outras variáveis importantes são: *helloSeparator* guarda o valor do byte que equivale a ':' e *randomBuffer* que é responsável por guardar o valor do número aleatório gerado para ser utilizado como desafio ao servidor OpenID. A variável *dataInstalled*, que é inicializada como falso durante a instalação e muda de valor uma única vez, quando a operação de registro foi concluída. Essa variável impede que valores como a chave pública do servidor, sejam reinicializadas.

```
29
30  byte[] userId;
31  byte[] cardId;
32
33  byte[] helloSeparator;
34
35  byte[] randomBuffer;
36
37  boolean dataInstalled;
```

O construtor da aplicação é chamado uma única vez, durante a sua instalação. Esse construtor deve alocar todos os objetos persistentes aqui. Os objetos são alocados e corretamente inicializados. O par de chaves RSA do cartão são gerados aqui.

```
38  private JCCipher() {
39   //1) Allocate the Card's Private and Public Key
40   KeyPair kp = new KeyPair(KeyPair.ALG_RSA_CRT, KeyBuilder.
         LENGTH_RSA_1024);
41   kp.genKeyPair();
42
43   rsa_priv_key = (RSAPrivateCrtKey )kp.getPrivate();
44   rsa_pub_key = (RSAPublicKey)kp.getPublic();
45
46   rsa_pub_host_key = (RSAPublicKey)kp.getPublic();
47
48
49   //2) Allocate a buffer byte array with the predefined size
50   //for the APDU communication
51   buffer = new byte[MAX_APDU_MSG_SIZE_IN_BYTES];
52
53   userId = new byte[MAX_APDU_MSG_SIZE_IN_BYTES];
54   cardId = new byte[MAX_APDU_MSG_SIZE_IN_BYTES];
55
56   dataInstalled = false;
57
58   helloSeparator = new byte[1];
59   helloSeparator[0] = (byte)(':');
60
61   randomBuffer = new byte[RANDOM_SIZE];
62
63   cipher= Cipher.getInstance(Cipher.ALG_RSA_PKCS1, false);
64  }
```

A aplicação é instalada no método *install*. Esse método estático aloca um novo objeto JCCipher e registra o seu AID.

```
65
66  //Install Method for this Applet
67  public static void install(byte bArray[], short bOffset, byte bLength)
68    throws ISOException {
69   new JCCipher().register();
70  }
```

O método *process(APDU arg0)* é o principal da classe, cada requisição APDU feita pelo computador para a aplicação é processada e respondida nesse método. Primeiro, verifica-se que não é uma APDU de seleção (uma vez que o comando de seleção também chama o método *process*), em seguida, de acordo com o valor do campo em *instruction* (INS) da requisição APDU, trata-se o comando de uma forma. O valor da requisição APDU é guardada na variável "apdu".

```
71
72  public void process(APDU arg0) throws ISOException {
73   //If we've received the selection command we return 0x9000 as SW
74   if(selectingApplet())
75    return;
76
77   //The APDU byte array
78   byte[] apdu = arg0.getBuffer();
```

Os comandos de troca de chaves públicas durante o registro são definidos aqui. A variável *dataInstalled* evita que essas instruções sejam executadas mais de uma vez.

```
79
80    //Send JC public key's modulus
81    if(apdu[ISO7816.OFFSET_INS] == (byte) 0x40) {
82     if(!dataInstalled){
83      arg0.setOutgoing();
84      arg0.setOutgoingLength((short) ((KeyBuilder.LENGTH_RSA_1024)/8));
85      arg0.sendBytesLong(RSAPublicKeyMod,(short)0, (short) ((KeyBuilder.
            LENGTH_RSA_1024)/8));
86     }
87
88    //Send JC public key's exponent
89    } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x41) {
90     if(!dataInstalled){
91      arg0.setOutgoing();
92      arg0.setOutgoingLength(RSA_PUBLIC_EXP_SIZE);
93      arg0.sendBytesLong(RSAPublicKeyExp,(short)0, RSA_PUBLIC_EXP_SIZE);
94     }
95
96    //Receive the public key's modulus
97    } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x42) {
98     if(!dataInstalled){
99      short bytesRead = loadReceivedDataToBuffer(arg0);
100     rsa_pub_host_key.setModulus(buffer, (short) 0, bytesRead);
101    }
102
103   //Receive the public key's exponent
104   } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x43) {
105    if(!dataInstalled){
106     short bytesRead;
107     bytesRead = loadReceivedDataToBuffer(arg0);
108     rsa_pub_host_key.setExponent(buffer, (short) 0, bytesRead);
109    }
```

Comandos utilizados para testar as chaves da mensagem. Esses comandos não são mais necessários.

```
110
111   //Encrypt a received message with our private key and send it back
112   } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x44) {
113    short bytesRead = loadReceivedDataToBuffer(arg0);
114    cipher.init(rsa_priv_key, Cipher.MODE_ENCRYPT);
115    //The output buffer may be the same as the input one
116    cipher.doFinal(buffer, (short)0, bytesRead, buffer,(short)0);
117    arg0.setOutgoing();
118    arg0.setOutgoingLength(CIPHERED_TEXT_MAX_ARRAY_SIZE);
119    arg0.sendBytesLong(buffer,(short)0, CIPHERED_TEXT_MAX_ARRAY_SIZE);
120
121   //Decrypt a received message with our private key and send it back
122   } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x45) {
123    short bytesRead = loadReceivedDataToBuffer(arg0);
124    cipher.init(rsa_priv_key, Cipher.MODE_DECRYPT);
125    //The output buffer may be the same as the input one
126    cipher.doFinal(buffer, (short)0, bytesRead, buffer,(short)0);
127    arg0.setOutgoing();
128    arg0.setOutgoingLength(PLAIN_TEXT_MAX_ARRAY_SIZE);
129    arg0.sendBytesLong(buffer,(short)0, PLAIN_TEXT_MAX_ARRAY_SIZE);
130
131
132   //Encrypt a received message with our public key and send it back
```

```
133     } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x46) {
134      short bytesRead = loadReceivedDataToBuffer(arg0);
135      cipher.init(rsa_pub_key, Cipher.MODE_ENCRYPT);
136      //The output buffer may be the same as the input one
137      cipher.doFinal(buffer, (short)0, bytesRead, buffer,(short)0);
138      arg0.setOutgoing();
139      arg0.setOutgoingLength(CIPHERED_TEXT_MAX_ARRAY_SIZE);
140      arg0.sendBytesLong(buffer,(short)0, CIPHERED_TEXT_MAX_ARRAY_SIZE);
141
142     //Decrypt a received message with our public key and send it back
143     } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x47) {
144      short bytesRead = loadReceivedDataToBuffer(arg0);
145      cipher.init(rsa_pub_key, Cipher.MODE_DECRYPT);
146      //The output buffer may be the same as the input one
147      cipher.doFinal(buffer, (short)0, bytesRead, buffer,(short)0);
148      arg0.setOutgoing();
149      arg0.setOutgoingLength(PLAIN_TEXT_MAX_ARRAY_SIZE);
150      arg0.sendBytesLong(buffer,(short)0, PLAIN_TEXT_MAX_ARRAY_SIZE);
151
152
153     //Encrypt a received message with their public key and send it back
154     } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x48) {
155      short bytesRead = loadReceivedDataToBuffer(arg0);
156      cipher.init(rsa_pub_host_key, Cipher.MODE_ENCRYPT);
157      //The output buffer may be the same as the input one
158      cipher.doFinal(buffer, (short)0, bytesRead, buffer,(short)0);
159      arg0.setOutgoing();
160      arg0.setOutgoingLength(CIPHERED_TEXT_MAX_ARRAY_SIZE);
161      arg0.sendBytesLong(buffer,(short)0, CIPHERED_TEXT_MAX_ARRAY_SIZE);
162
163     //Decrypt a received message with their public key and send it back
164     } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x49) {
165      short bytesRead = loadReceivedDataToBuffer(arg0);
166      if(rsa_pub_host_key.isInitialized()){
167       cipher.init(rsa_pub_host_key, Cipher.MODE_DECRYPT);
168       //The output buffer may be the same as the input one
169       cipher.doFinal(buffer, (short)0, bytesRead, buffer,(short)0);
170       arg0.setOutgoing();
171       arg0.setOutgoingLength(PLAIN_TEXT_MAX_ARRAY_SIZE);
172       arg0.sendBytesLong(buffer,(short)0, PLAIN_TEXT_MAX_ARRAY_SIZE);
173      }
```

Os dois comandos abaixo transmitem respectivamente as identidades do usuário e do cartão, no servidor OpenID.

```
174
175     //Set the user identity
176     } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x4A) {
177      if(!dataInstalled){
178       useridLength = loadReceivedDataToBuffer(arg0);
179
180       JCSystem.beginTransaction();
181       Util.arrayCopy(buffer, (short)0, userId, (short)0, useridLength);
182       JCSystem.commitTransaction();
183      }
184
185     //Set the card ID
186     } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x4C) {
187      if(!dataInstalled){
188       cardidLength = loadReceivedDataToBuffer(arg0);
```

```
189
190       JCSystem.beginTransaction();
191       Util.arrayCopy(buffer, (short)0, cardId, (short)0, cardidLength);
192       JCSystem.commitTransaction();
193     }
```

O comando abaixo encerra o registro do cartão, definindo a variável *dataInstalled* permanentemente como "true".

```
194
195   //Finish installation and set dataInstalled to true
196   } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x50) {
197     dataInstalled = true;
```

O comando abaixo é responsável por gerar um número aleatório que servirá de desafio para o servidor OpenID. Ele deve ser enviado juntamente com o número de cadastro do usuário e desse cartão.

```
198
199   //Prepare hello message with the following format:
200   //<userID:cardID:random_number>
201   } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x51) {
202     RandomData random = RandomData.getInstance(RandomData.
          ALG_SECURE_RANDOM);
203     random.generateData(randomBuffer,(short)0,RANDOM_SIZE);
204
205     JCSystem.beginTransaction();
206     Util.arrayCopy(userId,(short)0,buffer,(short)0,useridLength);
207
208     Util.arrayCopy(helloSeparator,(short)0,buffer,(short)useridLength,(
          short)(1));
209
210     Util.arrayCopy(cardId,(short)0,buffer,(short)(useridLength+1),(short
          )(cardidLength));
211
212     Util.arrayCopy(helloSeparator,(short)0,buffer,(short)(cardidLength+
          useridLength+1)
213       ,(short)(1));
214
215     Util.arrayCopy(randomBuffer,(short)0,buffer,(short)(cardidLength+
          useridLength+2)
216       ,(short)(RANDOM_SIZE));
217     JCSystem.commitTransaction();
```

O comando abaixo decriptografa o desafio recebido pelo servidor OpenID e criptografa com a chave pública desse servidor.

```
219
220   //Respond to the servers challenge
221   } else if(apdu[ISO7816.OFFSET_INS] == (byte) 0x52) {
222     short bytesRead = loadReceivedDataToBuffer(arg0);
223     cipher.init(rsa_priv_key, Cipher.MODE_DECRYPT);
224     //The output buffer may be the same as the input one
225     cipher.doFinal(buffer, (short)0, bytesRead, buffer,(short)0);
226
227     cipher.init(rsa_pub_host_key, Cipher.MODE_ENCRYPT);
228     //The output buffer may be the same as the input one
229     cipher.doFinal(buffer, (short)0, CHALLENGE_SIZE, buffer,(short)0);
230
231     arg0.setOutgoing();
```

```
232     arg0.setOutgoingLength(CIPHERED_TEXT_MAX_ARRAY_SIZE);
233     arg0.sendBytesLong(buffer,(short)0, CIPHERED_TEXT_MAX_ARRAY_SIZE);
234
235   }
```